НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Фасолька М. О.
Завідувач кафедри
_____ Олександр Коваль
« __»__ _____2020 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: Модернізація програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників

Виконав:
студент IV курсу, групи ТМ-61
Фасолька Максим Олександрович _____

Керівник:
Професор кафедри АПЕПС, доктор технічних наук,
Отрох Сергій Іванович _____

Рецензент:
Доцент, кандидат технічних наук, Доцент кафедри ТК
Зенів Ірина Онуфріївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Фасолька Максим Олександрович
(прізвище, ім’я, по батькові)

1. Тема роботи Модернізація програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників

керівник роботи д.т.н Отрох С.І.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ”25” травня 2020р. № 1168-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи : персональний комп’ютер під керуванням операційної системи Microsoft Windows, мови програмування Python, JavaScript.

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) _ проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, зробити висновки за результатами роботи

5. Орієнтований перелік ілюстративного матеріалу

Візуалізація монолітної архітектури проти мікросервісної, Шаблон MVT, Структура шаблону, Архітектура SQLite без сервера, Загальний огляд архітектури докера

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”__”_____2020р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	Захист програмного продукту		
7.	Передзахист		
8.	Захист		

Студент _____ Фасолька М.О.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Отрох С.І.
(підпис) (прізвище та ініціали,)

ВІДГУК

керівника дипломної роботи

освітньо-кваліфікаційного рівня „бакалавр”

виконаної на тему : __Модернізація програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників

студентом _____Фасолькою Максимом Олександровичем_____

(прізвище, ім’я, по батькові)

Дипломна робота Фасольки М. О. присвячена модернізації програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників.

Виконана дипломна робота повністю відповідає поставленому завданню. Поставлені завдання студента виконано з успіхом, проявивши самостійність та наполегливість. Якість виконання роботи засвідчив високий рівень володіння сучасними системними та інформаційними технологіями. Виконаний програмний продукт дозволяє проводити моніторинг наносупутників КПІ та наповнювати систему різноманітним контентом.

Пояснювальна записка виконана з дотриманням стандартів, її зміст включає аналіз предметної області, опис методів виконання поставлених задач та опис програмної реалізації.

Робота виконана у повному обсязі та відповідає завданню і вимогам, що висуваються до бакалаврських робіт, а студент присвоєння відповідної кваліфікації - бакалавр з комп’ютерних наук та інформаційних технологій з напряму підготовки 122 Комп’ютерні науки та інформаційні технології, за спеціалізацією Інформаційні технології моніторингу довкілля.

Керівник дипломної роботи

___Професор, д.т.н __ _____ Отрох С.І.

(посада, вчені звання, ступінь) (підпис) (ініціали, прізвище)

РЕЦЕНЗІЯ

на дипломну роботу

освітньо-кваліфікаційного рівня „бакалавр”

виконаної на тему : __Модернізація програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників

студентом _____Фасолькою Максимом Олександровичем_____

(прізвище, ім’я, по батькові)

Представлена на рецензію дипломна робота студента Фасольки М. О. присвячена модернізації програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників.

Розроблена під час виконання дипломної роботи програмна система надає простий, зрозумілий функціонал та інтерфейс для моніторингу.

В пояснювальній записці наведено огляд додатку та описані технології, які використовувались для його реалізації. Також представлені проблеми з якими можна зіткнутись при роботі з хмарним середовищем. Робота містить пояснювальну записку в об'ємі 70 сторінок.

Робота виконана на високому технічному рівні та відповідає вимогам, які пред’являються до сучасних програмних продуктів. Робота заслуговує оцінки “відмінно”, а студент Фасолька М. О. заслуговує на присвоєння кваліфікації - бакалавр комп’ютерних наук та інформаційних технологій з напряму підготовки 122 Комп’ютерні науки та інформаційні технології, за спеціалізацією Інформаційні технології моніторингу довкілля.

Рецензент

Доцент, к.т.н.	_____	Зенів І.Ю.
(посада, вчені звання, ступінь)	(підпис)	(ініціали, прізвище)

АНОТАЦІЯ

Мета роботи — дослідження мікросервісних систем та їх застосування для системи моніторингу технічного стану та режиму функціонування наносупутників. Для розробки програмного забезпечення системи моніторингу була використана мова програмування Python з фреймворком Django. Розробка графічного інтерфейсу користувача відбувалась на основі Java Script, HTML5 та SCSS.

Розроблений система забезпечує можливість моніторингу технічного стану та режиму функціонування наносупутників.

Записка містить 78 сторінок, 13 рисунків, 3 додатоки і 11 посилань.

Ключові слова: мікросервісна архітектура, система моніторингу технічного стану та режиму функціонування, розподілені системи.

ABSTRACT

This thesis is devoted to the development of intellectual agent for microservice systems and their application for a system for monitoring the technical condition and operating mode of nanosatellites.

To develop the monitoring system software, I used the Python programming language with the Django framework. The development of a graphical user interface was based on JavaScript, HTML5, and SCSS.

The developed system provides the ability to monitor the technical condition and operation mode of nanosatellites.

Note note 78 pages, 13 figures, 3 document and 13 poseyn.

Key words: microservice architecture, system for monitoring the technical condition and operating mode, distributed systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DevOps - DEVelopment OPeration

API - Application Programming Interface

HTTP - Hypertext Transfer Protocol

SOAP - Simple Object Access Protoco

RPC - Remote procedure call

REST - Representational state transfer

AMQP - Advanced Message Queuing Protocol

MVC - Model view controller

ORM - Object-relational mapping

TCP/IP – Transmission Control Protocol/Internet Protocol ACID

DOM - Document Object Model

CVS - Concurrent Versions System

UI - User interface

HCI - Human–computer interaction

UX - User Experience

UCD - User-centered design

CRM - Customer relationship management

CMS - Content management system

SEO - Search engine optimization

CLI - Command-line interface

ВСТУП

Нові відкриття завжди стимулювались потребою оптимізації робочого процесу для підвищення ефективності роботи. Так, для ефективної розробки наносупутників важливо постійно стежити за їх роботою. Система моніторингу потребує даних про поточний стан всіх частин системи для ефективного реагування та подальшого виправлення недоліків у роботі системи.

Одним із рішень цієї проблеми є мікросервісна архітектура. Це підхід до створення систем, який з'явився в контексті таких практик DevOps, як безперервна інтеграція, безперервна доставка і віртуалізація / контейнеризація. Він орієнтований на реалізацію декількох невеликих автономних програм або «сервісів», які взаємодіють один з одним для надання складної системи або додатку. Цей підхід відрізняється від більш традиційних підходів, таких як монолітні додатки, де одна кодова база містить кілька функцій, розгорнутих як єдине ціле. Моноліти з часом стають проблематичними, так як додаються нові функції. Кодова база зростає і може стати фрагментованою, погано організованою і схильною до проблемних місць.

На відміну від монолітів, мікросервіси ділять ціле додаток / послугу на окремі маленькі програми, які служать одній меті або функціональності незалежно. Складніша функціональність досягається шляхом об'єднання частин разом.

Такий підхід до створення системи дозволяє проводити ефективний моніторинг та розвиток сфери в цілому. Тому для вирішення проблеми було запропоновано створити систему моніторингу, як частину мікросервісної архітектури додатку. Така система має відображати інформацію про наносупутники в реальному часі.

Було поставлено завдання: проаналізувати мікросервісну архітетуру, розробити та спроектувати архітектурні шляхи вирішення для даної системи, розробити веб-систему, яка відображатиме всі можливі дані про супутники та зможе розвивати сферу побудови наносупутників в КПІ .

Для розробки програмного забезпечення системи моніторингу була використана мова програмування Python з фреймворком Django. Розробка графічного інтерфейсу користувача відбувалась на основі Java Script, HTML5 та SCSS.

1 АНАЛІЗ ПРОБЛЕМИ СТОВОРЕННЯ СИСТЕМИ МОТІНОРИНГУ ТЕХНІЧНОГО СТАНУ ТА РЕЖИМУ ФУНКЦІОНУВАННЯ НАНОСУПУТНИКІВ

Сучасні технології відіграють одну з провідних ролей в повсякденному житті, а Інтернет розвивається стрімко і сьогодні більшість компаній вже мають свої сайти. Існує багато варіантів їх створення та одним з найцікавіших є ідея веб системи.

Веб система моніторингу технічного стану та режиму функціонування наносупутників — система, що забезпечує процес моніторингу за наносупитниками КПІ. Вона дає змогу визначити поточне положення на орбіті, висоту, швидкість, азимут, нахил, кут повернення, похил та супутниковий період супутників.

Наразі існує багато сайтів конкурентів в тому числі іноземних, що пропонують системи моніторингу, такі як: cubesat.org, sputnix.ru, aac-clyde.space та інші. Кожний з них має контент, що розповідає про їх можливості розробки, новини пов’язані з компаніями та сферою супутників. Здебільшого це статичні сайти, що не відображають повної картини розвитку та оновлюються дуже рідко. Таке рішення не є прийнятним для нашої системи моніторингу. Такі системи не дозволяють повністю розкрити потенціал сфери та й проводити моніторинг вцілому. Також вони дуже стислі і забезпечують малий функціонал.

У перелічених системах немає можливості визначити період, висоту та швидкість, що допомагають в подальших розрахунках та дослідженнях. Також системи не дають змогу залучати більше аудиторії, публікувати нові статті та новини.

1.1 Опис архітектурних рішень

Мікросервісна архітектура - це підхід до створення систем, який з'явився в контексті таких практик DevOps, як безперервна інтеграція, безперервна доставка і віртуалізація / контейнеризація. Він орієнтований на реалізацію декількох невеликих автономних програм або «сервісів», які взаємодіють один з одним для надання

складної системи або додатку.

Мікросервіси запозичують зі старої філософії розробки програмного забезпечення: програми повинні бути спрямовані на те, щоб робити щось одне і робити це добре. Цей підхід відрізняється від більш традиційних підходів, таких як монолітні додатки, де одна кодова база містить кілька функцій, розгорнутих як єдине ціле. Моноліти з часом стають проблематичними, так як додаються нові функції. Кодова база зростає і може стати фрагментованою, погано організованою і схильною до вузьких місць.

На відміну від монолітів, мікросервіси ділять ціле, додаток / послугу на окремі маленькі програми, які служать одній меті або функціональності. Окремі компоненти менше за розміром, їх легше обслуговувати, і їх можна масштабувати незалежно. Складніша функціональність досягається шляхом об'єднання частин разом. Приклад таких архітектурних рішень можемо побачити на рисунку 1.1.

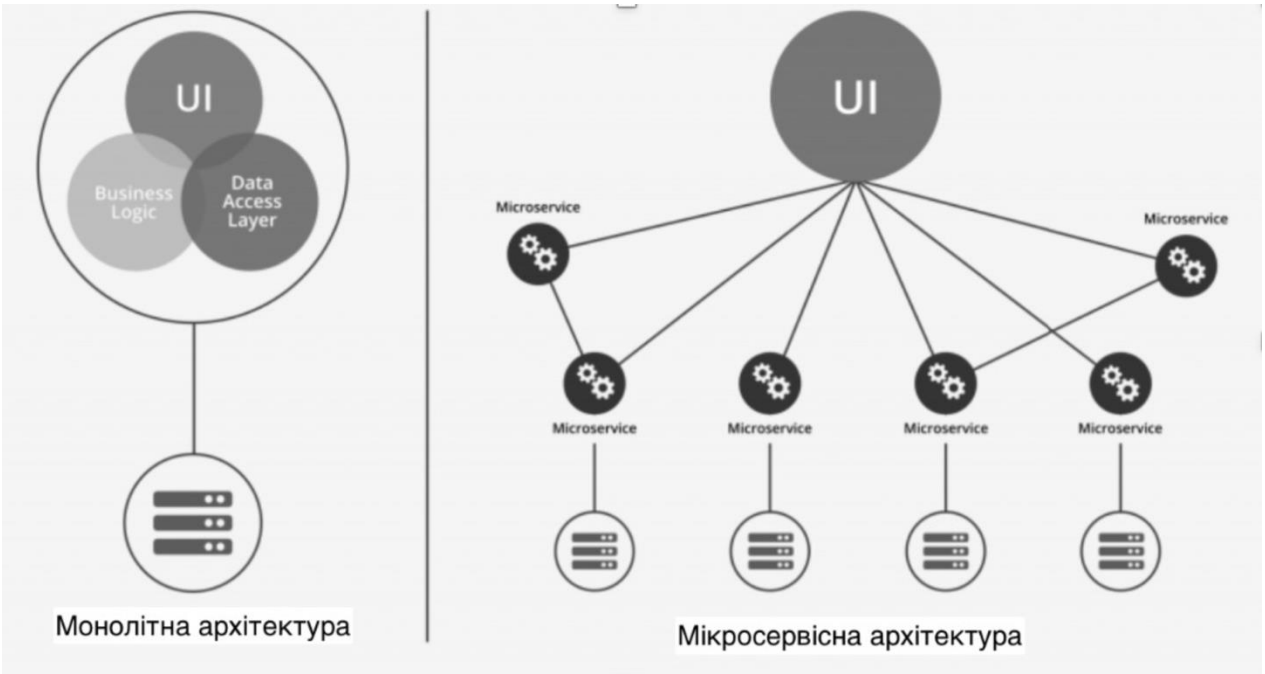

Рисунок 1.1 – Візуалізація монолітної архітектури проти мікросервісної

Веб-додатки унікальні тим, що їх кодові бази вже мають природну сегментацію в залежності від того, де повинен виконуватися код. Розглянемо рисунок 1.1, який показує три частини:

Інтерфейс - інтерфейс на стороні клієнта, зазвичай написаний на HTML, CSS або JavaScript

Рівень доступу до даних - база даних, в якій зберігається стан додатку.

Бізнес логіка - логіка на стороні сервера, яка взаємодіє з базою даних, може надавати API для взаємодії з даними або керувати рендерингом клієнтського інтерфейсу на стороні сервера.

Монолітні додатки об'єднують три компонента в один логічний виконуваний файл. Ця структура мала сенс при початковій підготовці програми, оскільки дозволяла групувати загальні функції без надмірності і забезпечувала простий спосіб управління розгортанням. Всі функції програми виконуються в рамках одного процесу, а функції поділяються з використанням різних функцій мови програмування, на якому написано додаток.

У міру зростання моноліту застосування незмінно ускладнюється. У міру додавання нового коду потрібен багато часу, щоб організувати функціональність, щоб не було конфліктуючих проблем або непотрібного дублювання коду. Розростання коду ускладнює виправлення помилок або реалізацію нових функцій.

Згодом виникають інші проблеми:

- Цикли змін пов'язані один з одним - оскільки моноліти тісно об'єднують три компоненти додатку, оновлення призначеного для користувача інтерфейсу вимагає тестування, налагодження та розгортання всіх компонентів, включаючи логіку бази даних і сервера. Відновлення та розгортання всього монолітного додатку займає дуже багато часу і створює часті випуски.
- Ізоляція компонентів є складним завданням. Підтримка модульності різних компонентів вимагає продуманої роботи і планування. У більшості кодових баз часто існує деякий збіг між модулями, і саме тоді

функціональність починає руйнуватися. Це ускладнює збереження змін, які впливають тільки на один модуль а починають впливати на інші.

- Компоненти тісно пов'язані - оскільки кожен компонент суворо прив'язаний до інших, зміни в окремих функціях можуть бути проблематичними протягом усього життєвого циклу програми. Якщо для певних частин додатка потрібно більше ресурсів, необхідно масштабувати весь додаток, а не тільки необхідні служби.
- Обслуговування системи - якщо відбувається збій в додатку, вся система повинна бути відключена, щоб виправити це. Це збільшує час простою служби, оскільки при усуненні проблем може знадобитися значна кількість часу для повторного тестування і повторного розгортання.

Мікросервіси усувають проблеми монолітів, розділяючи компоненти кодової бази на набори дрібніших процесів, які можуть взаємодіяти зі службами за допомогою чітко визначеного інтерфейсу. Кожен мікросервіс заснований на бізнес-цілях для всієї програми і призначений для однієї мети. Потім кожен сервіс виконується в своєму власному процесі, що дозволяє масштабувати його незалежно від іншої частини програми. Якщо необхідно відключити певну службу, цей єдиний процес може бути відключений для обслуговування, не переводячи всю систему в автономний режим.

1.2 Опис мікросервісної архітектури

Для проектування мікросервісної архітектури потрібно:

— Сервісні компоненти

— Правильний інструмент для роботи

— Організована бізнес-спроможність

— Децентралізоване управління даними

— Розумні кінцеві точки

Компоненти повинні бути в основному зосереджені на одному основному елементі функціональності. Кожен сервіс має свою власну кодову базу і працює в своєму власному процесі. Це відрізняє його від використання бібліотек в якості компонентів, оскільки поділ коду і функціональних можливостей забезпечується середовищем виконання і забезпечує ізоляцію, тоді як бібліотека в моноліті є частиною більш великого додатку, і вся система повинна бути переведена в автономний режим для повторного розгортання.

Друга важлива перевага, що надається групуванням компонентів в якості сервісів, полягає в тому, що виникає необхідність надати явний, опублікований інтерфейс, який потім використовують інші частини системи. Це зміцнює контракти API і в цілому поліпшує якість коду.

У традиційних командах розробників технічні групи часто розбиті на дисципліни. Це часто призводить до створення команди розробників бази даних, команди призначеного для користувача інтерфейсу, команди бекенда.

Організації, які суворо дотримуються стратегії мікросервісів, організовуватимуть команди на основі бізнес-можливостей. Замість групи, що працює з базами даних, може існувати група по електронній комерції, команда по контенту і т.д. Мета полягає в тому, щоб дозволити командам володіти своїми власними продуктами від концепції до реалізації і розгортання. Це забезпечує почуття причетності і часто призводить до отримання більш якісних продуктів.

Мікросервіси намагаються стримувати функціональність. Розглянемо різницю між протоколом, таким як HTTP, який можна використовувати для передачі практично будь-яких типів даних, в порівнянні з SOAP, який високо оптимізований для віддалених викликів процедур (RPC). Існує два загальних протоколи, які часто використовуються для мікросервісов (один синхронний, а інший асинхронний): HTTP (часто використовується для надання типу інтерфейсу REST) і легкі протоколи обміну повідомленнями, такі як розширений протокол черги повідомлень (AMQP) або протокол Apache Kafka.

Перевага поділу функціональності моноліту полягає в тому, що він дозволяє використовувати різні мови програмування або бібліотеки при створенні компонентів. Зв'язок відбувається за загальними протоколами, які підтримуються майже кожною мовою програмування, а це значить, що практично будь-який набір

інструментів може бути використаний для вирішення проблеми. Якщо потрібна обробка в реальному часі з дуже низькою затримкою, ці компоненти можуть бути написані на мові, такій як C ++. Якщо потрібно машинне навчання, це може бути реалізовано в Python. Для компонентів, пов'язаних з призначеним для користувача інтерфейсом, вони можуть бути реалізовані в JavaScript або в одному з його варіантів.

У монолітних додатках рішення по зберіганню даних часто засновані на вимогах найсуворіших компонентів системи без урахування того, як це може вплинути на інші компоненти. Наприклад, в додатку для продажу процеси пакетної аналітики, які можуть працювати з використанням щогодинних, щоденних або навіть щотижневих моментальних знімків даних, можуть виявитися вузькими місцями в системах, що вимагають з точністю до секунди.

У мікросервісах можна розділити такі проблеми і дозволити кожному компоненту працювати з даними, які йому необхідні. Пакетна аналітика може спиратися на архівні дані, а доступність визначається шляхом читання з потокової бази даних. Використання правильного джерела даних про програму звільняє базу даних для більш ефективної обробки запитів з систем реального часу, не надаючи надмірного тиску ні на одну з них.

Хоча мікросервіси надають багато переваг, вони також можуть принести складності, які не належать до моноліту. Оскільки додатки, керовані мікросервісами, являють собою розподілені системи, які обмінюються даними по мережі, їм доводиться мати справу з мережевою взаємодією. Моноліти або досягають успіху, або провалюються як єдине ціле, вони або доступні, або ні. Мікросервіси більш деталізовані, частина системи може бути працездатна, а інша не працює.

Оскільки служба може вийти з ладу без попередження, необхідно швидко виявляти і відновлювати простої. Аналогічно, система повинна бути здатна функціонувати, якщо пов'язаний компонент недоступний. В результаті команди, що управляють мікросервісами, часто роблять упор на моніторинг програм в режимі реального часу, де вони відстежують як продуктивність (наприклад, кількість запитів, які служба може обробляти в секунду, так і кількість часу, необхідну для виконання запиту), а також відповідні бізнес-показники.

Отже, мікросервіси пропонують альтернативу традиційному способу розробки монолітних додатків. Вони поділяють складні взаємодії великих додатків на окремі компоненти, які запускаються в своїх власних процесах і взаємодіють за стандартними протоколами. Це дає багато переваг, таких як поліпшена еластичність, децентралізоване управління даними, кращу відповідність між технологіями і завданнями впровадження, а також відмовостійкість.

Незважаючи на те, що вони можуть здатися складними для реалізації, переваги мікросервісів окупаються для складних корпоративних додатків. Вони не тільки можуть прискорити випуск нових функцій і забезпечити інтеграцію з більш надійним процесом (таким як безперервна інтеграція і безперервне розгортання), але і всім додатком буде легше керувати, створювати, тестувати і розгортати.

2 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Концепції мікросервісної архітектури можна застосувати для управління та моніторингу до різноманітних систем різного призначення. Зокрема, такий спосіб побудови системи можна успішно застосувати для системи моніторингу технічного стану та режиму функціонування наносупутників.

Тому для створення програмного продукту потрібно проаналізувати доцільність такого архітектурного рішення та обгрунтувати його в розробці системи.

2.1 Архітектура MVC

Моде́ль–вигляд–контро́лер (MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Архітектура MVC дозволяє розробникам змінювати візуальну частину програми та частину бізнес-логіки окремо, не впливаючи один на одного. Цю архітектуру в Django часто називають моделлю-виглядом-шаблоном (MVT). Три шари (Модель, Вид і Шаблон) відповідають за різні речі і можуть використовуватися незалежно(Рис.2.1.1).

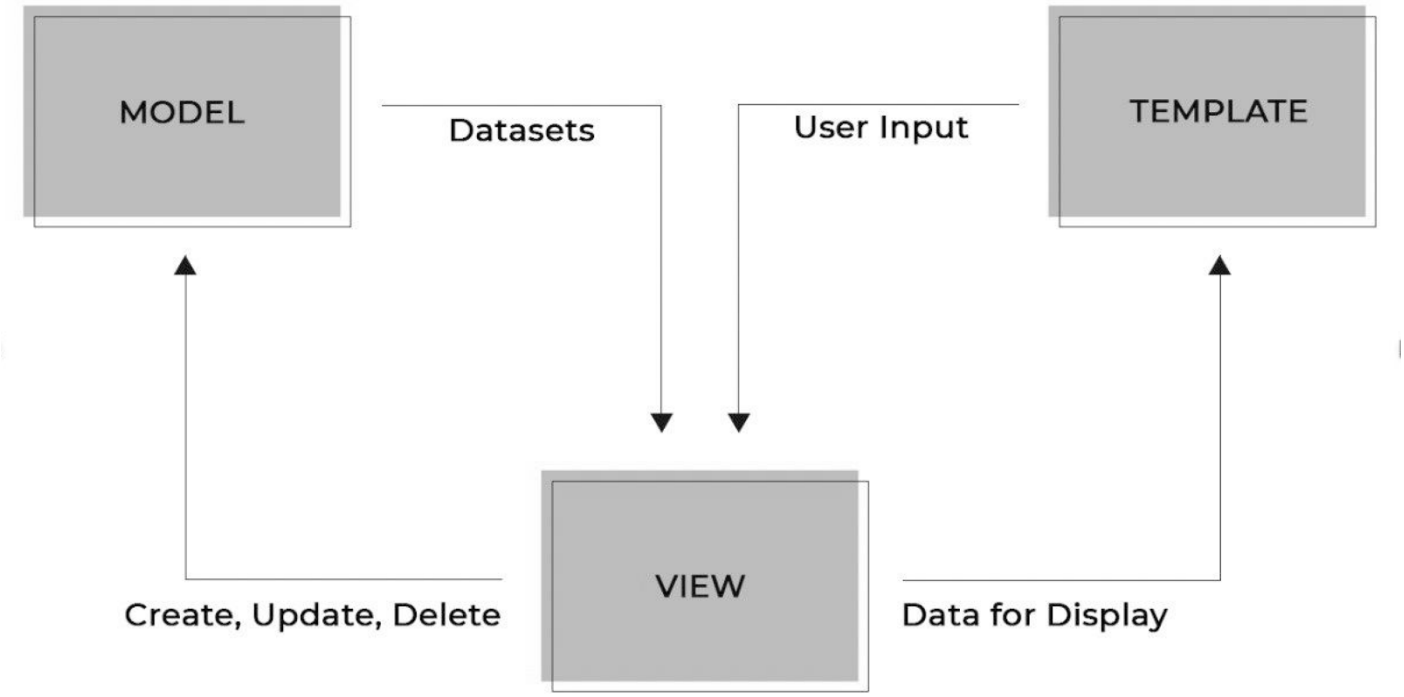

Рис. 2.1.1- шаблон MVT

Модель є єдиним, остаточним джерелом інформації про ваші дані. Він містить основні поля і поведінку даних, які ви зберігаєте. Як правило, кожна модель відображається в одну таблицю бази даних. Навряд чи є додаток без бази даних, і Django офіційно підтримує чотири: PostgreSQL, MySQL, SQLite і Oracle. Моделі містять інформацію про ваші дані і представлені атрибутами (полями). Оскільки модель є простим класом Python, вона нічого не знає про інші шари Django. Зв'язок між рівнями можливий тільки через інтерфейс прикладного програмування (API). Моделі містять бізнес-логіку, призначені для користувача методи, властивості і інші речі, пов'язані з маніпулюванням даними. Крім того, моделі дозволяють створювати, зчитувати, оновлювати і видаляти об'єкти (набори даних) у вихідній базі даних.

Вигляд виконує завдання дерева: воно приймає запити HTTP, застосовує бізнес-логіку, яка надається класами і методами Python, і надає відповіді HTTP на запити клієнтів. Іншими словами, вигляд отримує дані з моделі і надає кожному шаблону доступ до конкретних даних, які повинні бути відображені, або обробляє дані заздалегідь.

Django має потужний шаблонізатор і власну мову розмітки з безліччю інструментів. Шаблони - це файли з HTML-кодом, які використовуються для візуалізації даних. Вміст цих файлів може бути статичним або динамічним. Оскільки в шаблоні немає бізнес-логіки, він призначений тільки для представлення даних. Структура зображена на наступному рисунку 2.1.2 допомагає Django виконувати різні завдання.

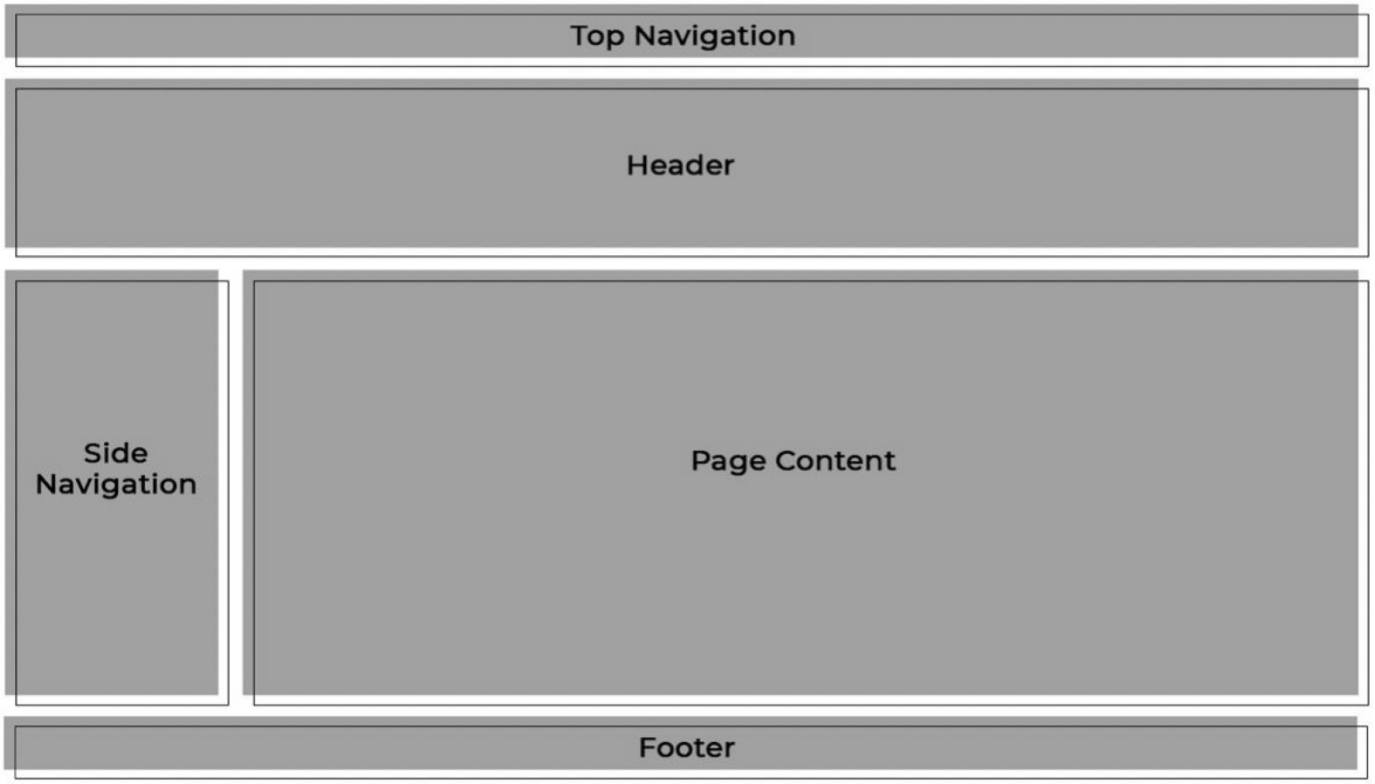

Рис. 2.1.2- структура шаблону

2.2 Django ORM

Django ORM цінується за своє об'єктно-реляційне відображення, яке допомагає розробникам взаємодіяти з базами даних. Об'єктно-реляційний вигляд (ORM) - це бібліотека, яка автоматично передає дані, що зберігаються в базах даних, таких як PostgreSQL і MySQL, в об'єкти, які зазвичай використовуються в коді програми.

Здатність Django ORM витягувати інформацію прискорює розробку веб-додатків і допомагає розробникам створювати робочі прототипи в найкоротші терміни. Розробникам не обов'язково знати мову, використовувану для взаємодії з базою даних для маніпулювання даними.

Крім того, Django ORM допомагає розробникам перемикатися між реляційними базами даних з мінімальними змінами коду. Це може дозволити використовувати SQLite для локальної розробки і, наприклад, переключитися на MySQL. Однак, як правило, найкраще використовувати одну базу даних, щоб уникнути помилок, які можуть виникнути під час переходу.

SQLite - це програмна бібліотека, що надає систему керування базами даних. “Lite” в SQLite означає легкий з точки зору налаштування, адміністрування бази даних і необхідних ресурсів.

Зазвичай для СУБД, таких як MySQL, PostgreSQL, потрібен окремий серверний процес для роботи. Додатки, які хочуть отримати доступ до сервера бази даних, використовують протокол TCP / IP для відправки та отримання запитів. Це називається архітектура клієнт / сервер. Наступна діаграма ілюструє архітектуру клієнт / сервер(Рис.2.2.1):

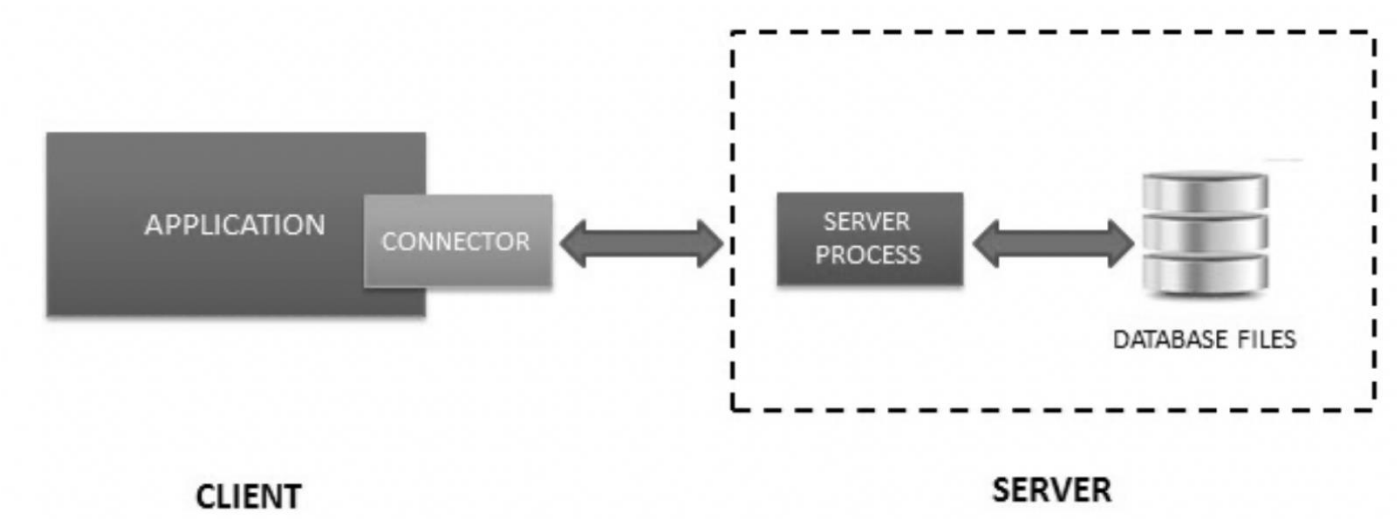

Рис. 2.2.1- архітектура клієнт / сервер

SQLite не працює таким чином і не вимагає запуску сервера. База даних SQLite інтегрована з додатком, який звертається до бази даних. Додатки взаємодіють з базою даних SQLite для читання і запису безпосередньо з файлів бази даних, що зберігаються на диску. Наступна діаграма ілюструє архітектуру SQLite без сервера(Рис.2.2.2):

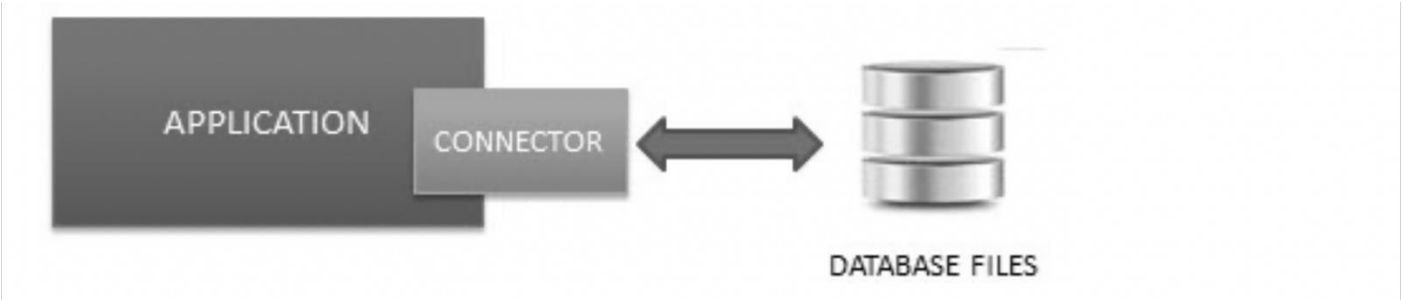

Рис. 2.2.2- архітектуру SQLite без сервера

SQLite є автономною, тобто вимагає мінімальної підтримки з боку операційної системи або зовнішньої бібліотеки. Через безсерверну архітектуру вам не потрібно встановлювати SQLite перед його використанням. Немає серверного процесу, який потрібно налаштувати, запустити і зупинити. Крім того, SQLite не використовує ніяких файлів конфігурації. Всі транзакції в SQLite повністю сумісні з ACID. Це означає, що всі запити і зміни є атомарними, послідовними, ізольованими і надійними. Іншими словами, всі зміни в транзакції відбуваються повністю або не відбуваються взагалі, навіть коли відбувається непередбачена ситуація, така як збій програми або збій операційної системи.

SQLite використовує динамічні типи для таблиць. Це означає, що ви можете зберігати будь-яке значення в будь-якому стовпці, незалежно від типу даних. Також вона дозволяє одному з'єднанню з базою даних одночасно звертатися до декількох файлів бази даних. Це приносить багато корисних функцій, таких як об'єднання таблиць в різних базах даних або копіювання даних між базами даних за допомогою однієї команди. SQLite здатний створювати бази даних в пам'яті, з якими дуже швидко працювати.

2.3 Контейнерна розробка

Контейнерна розробка - це робочий процес розробки, який пише, запускає і тестує код всередині контейнерного середовища. Контейнер - це упакований файл, який містить додаток (часто структурований як мікросервіс) разом з його вихідним кодом, бібліотеками, ресурсами і будь-якими іншими залежностями, необхідними для правильного виконання.

Ідея спрощення складного додатка до контейнера дає ряд переваг. Ці переваги включають в себе:

- послідовне розгортання і виконання
- переносимість додатку, коли один і той же артефакт може бути використаний при розробці, підготовці та запуску

- способи зберігання, транспортування та розгортання додатків в різних середовищах виконання

Контейнери вирішують ряд проблем в розробці програмного забезпечення. У традиційному робочому процесі розробка складатиметься з написання програми, побудови системи, тестування отриманих артефактів, а потім упаковки компонентів таким чином, щоб їх можна було розгорнути. Багато етапів цього процесу часто вимагають ручних взаємодій, які можуть займати багато часу і бути схильні до помилок.

Використання контейнерів усуває багато проблем, забезпечуючи певну формальність при складанні та упаковці додатку, створюючи єдиний артефакт, який можна використовувати для тестування, і надаючи погоджений артефакт, який можна використовувати при розробці, підготовці та виробництві. Оскільки весь процес забезпечує правильність того місця, де буде розгорнуто програму, проблеми і помилки, пов'язані з відмінностями в середовищі, зводяться до мінімуму. Крім того, автоматизовані процеси можуть бути побудовані навколо збірки, тестування і розгортання; що дозволяє випускати програмне забезпечення з більшою швидкістю.

Контейнери віртуалізують ресурси процесора, пам'яті, сховища і мережі на рівні операційної системи. Процеси, що виконуються всередині одного контейнера, не помітні процесам, що виконуються всередині іншого контейнера. Це дає перевагу: чутливі процеси можуть бути ізольовані в одному логічному вікні, невидимому для інших процесів, що виконуються на машині. Крім того, логічна ізоляція надає розробникам ізольоване уявлення ОС, ізольованою від іншого програмного забезпечення, яке також може бути розгорнуто. Ця ізоляція допомагає спростити середовище виконання і запобігти ненавмисній взаємодії між схожими програмами, які можуть використовувати один і той же хост.

Поширеною причиною програмних помилок є відмінності в середовищі. Розробник може мати один набір залежностей, встановлених на локальній робочій станції (часто накопичуються за місяці або роки роботи над додатком), підготовка може мати трохи інший набір, а виробництво - третій набір. Крім того, коли код виконується, файли кешу генеруються і стають частиною середовища і можуть

сприяти тонким взаємодіям, які викликають важко діагностуючі помилки.

Хоча контейнери і їх шаблони можуть бути створені в інтерактивному режимі шляхом внесення змін у вигляді «зображення», набагато частіше їх збирають з використанням так званого Dockerfile. Dockerfiles можуть посилатися на інші образи контейнерів, а потім розширювати їх додатковою конфігурацією і компонентами. Оскільки файл містить точну специфікацію контейнера, можна підтвердити, що всі «зображення», створені з одного і того ж Dockerfile, будуть працювати однаково. Образи можуть бути версійними, що означає, що можна точно визначити компоненти складного додатка, що охоплює кілька контейнерів, і гарантувати, що всі компоненти працюють так, як очікується.

Створення нових зображень контейнерів часто автоматизовано, це дозволяє створювати більші процеси, такі як безперервна інтеграція та конвеєри розгортання, які можуть значно прискорити розгортання нових версій програмного забезпечення після їх ретельного тестування.

Контейнерна розробка не тільки підвищує продуктивність праці розробників, але й допомагає організаціям стандартизувати операції і процеси. Коли додатки упаковані в контейнери, операційним командам стає набагато простіше переміщати перевірену версію програми в проміжну, а потім у виробничу область. Запуск і запис для кожного контейнера узгоджені, тобто ви знаєте, що контейнер буде форматувати, розподіляти ресурси, передавати значення конфігурації і управляти артефактами кожен раз однаково. Це дозволяє автоматизувати ключові частини конвеєра розгортання і дозволяє швидко випускати програмне забезпечення.

3 АНАЛІЗ ТА ОБҐРУНТУВАННЯ РЕАЛІЗАЦІЇ

3.1 Оцінка вимог

Проаналізувавши архітектуру системи та її функції було сформульовано головні вимоги:

- написання статей для інформування про новини космосу та розвитку цієї сфери ;
- створення контенту для пристроїв, використовуючи макети на основі стовпців, тексту, кнопок, зображень, відео, листівок, каруселі та багато іншого з потужною системою сітки Foundation для сайтів;
- публікування оновлення, створення нових сторінок;
- планування часу публікацій, коли вони з'являться на сайті;
- завантаження зображення з високою роздільною здатністю і встановлення фокусу;
- завантаження відео контенту;
- перегляд місцезнаходження супутника;
- визначення супутникового періоду;
- перегляд його координат, висоти, швидкості, швидкості обертання.

В результаті розробки програмного продукту, користовачу буде надана можливість будувати візуальну частину сайту та повністю змінювати її під його потреби. За допомогою послідовності «блоків» контенту, які можуть бути перебудовані в міру необхідності, можна створювати безліч різноманітного контенту та наповнювати ним сайт.

3.2 Веб-компоненти

В сучасній веб-розробці є ряд ключових компонентів, які виникли в результаті галузевих змін за останнє десятиліття. Деякі з них - інструменти, деякі - ідеї і

підходи. Всі вони засновані на необхідності управляти балансом складних вимог з продуктивністю і простотою.

Веб-компоненти є найбільш важливим елементом, якщо ми намагаємося охарактеризувати «сучасну» веб-розробку. Принцип простий: веб-компоненти надають набір повторно використовуваних для користувача елементів. Це полегшує створення веб-сторінок і додатків без написання додаткових рядків коду, які ускладнюють вашу кодову базу. Тут важливо пам'ятати, що веб-компоненти покращують інкапсуляцію. Ця концепція, яка насправді про побудову в більш модульному і слабкому зв'язаному вигляді, має вирішальне значення, коли ми думаємо про те, що робить сучасну веб-розробку сучасною. Є три основних елементи веб-компонентів:

- Призначені для користувача елементи, які представляють собою набір API-інтерфейсів JavaScript, які ви можете викликати і визначати, як вам потрібно для їх роботи.
- Тіньовий DOM, який діє як DOM, прикріплений до окремих елементів на вашій сторінці. Це істотно ізолює ресурси, необхідні для роботи на вашій сторінці різних елементів і компонентів, що спрощує управління з точки зору розробки і може підвищити продуктивність для користувачів.
- HTML-шаблони, що представляють собою фрагменти HTML-коду, які можна використовувати повторно і викликати тільки при необхідності.

Ці елементи разом малюють картину сучасної веб-розробки, в якій розробники намагаються впоратися з більшою складністю і витонченістю, одночасно підвищуючи свою продуктивність і ефективність

Якісний додаток - це додаток, розроблений з урахуванням нинішніх і майбутніх потреб. Додаток розроблений для задоволення довгострокових потреб користувача, зміни технологій, який може масштабуватися в міру зростання і бути легко обслуговуваним.

Для створення якісного додатку потрібне детального розуміння сучасних принципів розробки. Тож при розробці програмного продукту були застосовані

принципи об’єктно-орієнтованого програмування.

Об'єктно-орієнтоване програмування (ООП) - це не що інше, як те, що дозволяє писати програми за допомогою певних класів і об'єктів реального часу. Можна сказати, що цей підхід дуже близький до реального світу і його додатків, тому що стан і поведінку цих класів і об'єктів практично збігаються з об'єктами реального світу.

Об’єктно-орієнтоване програмування базується на наступних концепціях:

- інкапсуляція
- абстракція
- поліморфізм
- успадкування

Інкапсуляція - це механізм, який об'єднує дані (змінні екземпляра) і код, який діє на дані (методи), в єдине ціле, наприклад клас. Основне призначення інкапсуляції - повний контроль над даними за допомогою коду. У інкапсуляції змінні класу можуть бути приховані від інших класів і доступні тільки через методи їх поточного класу. Тому інкапсуляцією часто називають приховування даних. Інкапсуляція може бути описана як захисний бар'єр, який запобігає випадковому доступу коду і даних до іншого коду, визначеного поза класом. Доступ до даних і коду строго контролюється класом. Користувач повинен виконувати лише обмежений набір операцій над прихованими членами класу, виконуючи спеціальні функції, зазвичай звані методами.

Абстракція - це процес приховування деталей реалізації від користувача, користувачеві надається тільки функціональність. Іншими словами, у користувача буде інформація про те, що робить об'єкт, а не про те, як він це робить. Завдяки абстракції ми можемо приховувати складні речі всередині класів і створюючи відкриті методи для доступу до них.

Успадкування може бути визначено як процес, в якому один клас набуває властивостей (методи і поля) іншого. В об'єктно-орієнтованому програмуванні успадкування - це коли об'єкт або клас грунтується на іншому об’єкті або класі, використовуючи ту ж реалізацію для підтримки такої ж поведінки. Існує декілька типів успадкування: єдине успадкування, множинне успадкування, багаторівневе

спадкування ,ієрархічне успадкування, гібридне успадкування.

Переваги успадкування полягають в наступному. Успадкування сприяє повторному використанню. Коли клас успадковує інший клас, він може отримати доступ до всіх функцій успадкованого класу. Можливість повторного використання підвищує надійність. Код базового класу буде вже протестований і налагоджений. Оскільки існуючий код використовується повторно, це призводить до зниження витрат на розробку та обслуговування. Успадкування змушує підкласи слідувати стандартного інтерфейсу. Також воно допомагає зменшити надмірність коду і підтримує розширюваність коду.

Незважаючи на все вищесказане є й недоліки. Вони полягають в наступному:

- Успадковані функції працюють повільніше, ніж звичайні функції, так як існує опосередкованість.
- Неправильне використання успадкування може привести до неправильних рішень.
- Часто члени даних в базовому класі залишаються невикористаними, що може привести до втрати пам'яті.
- Спадкування збільшує зв'язок між базовим класом і похідним класом тому зміна базового класу вплине на всі дочірні класи.

В об'єктно-орієнтованому програмуванні поліморфізм (від грецького значення «мати кілька форм») є характеристикою здатності призначати різне значення або використання будь-чого в різних цілях.

Завдяки цим концепціям ООП гнучке та зручне. Веб-додаток, розроблений за цими концепціями є відображенням предметної області як ієрархічних об’єктів.

4 ЗАСОБИ РОЗРОБКИ

Важливим чинником, під час розробки програмного продукту, є вибір засобів програмної реалізації та технологій. Середовищем розробки інформаційної системи було обрано IntelliJ IDEA PyCharm. Для створення графічного інтерфейсу системи моніторингу використовувався веб-інтерфейс, з використанням JavaScript, SCSS, HTML5 та бібліотеки Backbone.js.

Для розробки логіки та моделей використовувалась об’єктно-орієнтована мова програмування Python з використанням фреймворку Django. Всередині і зовні система опирається на Wagtail - це безкоштовна система управління контентом (CMS), доступна для Django.

4.1 Середовище розробки IntelliJ IDEA

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія «Ultimate Edition» відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA як базис для своєї нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android.

PyCharm - інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладчик, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django. PyCharm розроблена компанією JetBrains на основі IntelliJ IDEA.

PyCharm - це крос-платформне середовище розробки, яке сумісне з Windows, MacOS, Linux. PyCharm Community Edition (безкоштовна версія) знаходиться під ліцензією Apache License, а PyCharm Professional Edition (платна версія) є пропрієтарним ПО.

4.2 Веб-інтерфейс

Веб-інтерфейс - це сторінка, з якою користувачі взаємодіють, коли сайт повністю завантажується через веб-браузер. Веб-сайт являє собою набір коду, але цей код не підходить для взаємодії з користувачем. Щоб гарантувати, що відвідувачі можуть використовувати сайт, код повинен мати веб-інтерфейс, з яким користувачі можуть легко взаємодіяти.

Багато в чому велика частина веб-дизайну пов'язана зі створенням веб-інтерфейсу, який дозволяє відвідувачам користуватись сайтом так, щоб це було корисно.

З цієї причини важливо створити веб-інтерфейс, який буде привабливим і інтуїтивно зрозумілим. Користувачі повинні бути в змозі знайти інформацію, яку вони шукають. Цю нішу часто називають дизайном користувацького інтерфейсу.

Користувацький інтерфейс (UI) - це точка взаємодії людини і комп'ютера в зв'язці з пристрями. Це може включати екран дисплея, клавіатуру, мишку і зовнішній вигляд робочого столу. Це також спосіб взаємодії користувача з додатком або веб-сайтом. Зростаюча залежність багатьох підприємств від веб-додатків і мобільних додатків спонукала багато компаній приділяти підвищену увагу призначену для користувача інтерфейсу, прагнучи поліпшити загальне враження користувача. Такі веб-сайти, як Airbnb, Dropbox та Virgin America, демонструють потужний дизайн інтерфейсу користувача. Компанії створили приємні, легко функціонуючі, орієнтовані на користувача та його потреби дизайни.

Про UI часто говорять в поєднанні з призначеним для користувача інтерфейсом (UX), який може включати в себе естетичний зовнішній вигляд пристрою, час відгуку і контент, який надається користувачу в контексті призначеного для користувача інтерфейсу. Обидва терміни підпадають під концепцію взаємодії людини з комп'ютером (HCI), яка є областю дослідження, зосередженого на створенні комп'ютерних технологій і взаємодії між людьми і всіма формами IT-дизайну. Зокрема, HCI вивчає такі області, як UCD, дизайн UI і дизайн UX.

4.3 Django Framework

Django - це широко використовувана платформа веб-розробки високого рівня з відкритим вихідним кодом. Він надає безліч можливостей для розробників "з коробки", тому розробка може бути швидкою. Однак веб-сайти, створені на його основі, є захищеними і масштабованими одночасно.

Django, за великим рахунком, є основним веб-середовищем для розробників на Python в наші дні, і неважко зрозуміти чому. Він відрізняється тим, що приховує багато логіки конфігурацій і дозволяє зосередитися на можливості швидко і масштабно будувати.

Django позиціонує себе як веб-фреймворк Python високого рівня, який сприяє

швидкій розробці і чистому, прагматичному дизайну. Створений досвідченими розробниками, він бере на себе більшу частину турбот веб-розробки, тому можна зосередитися на написанні свого застосунку без необхідності заново винаходити колесо.

Django надає практично все, що розробники можуть захотіти зробити «з коробки». Оскільки все, що вам потрібно, є частиною єдиного «продукту», все це бездоганно працює разом, відповідає послідовним принципам проектування і має велику і актуальну документацію.

Django може бути використаний для створення практично будь-якого типу веб-сайтів - від систем управління контентом до соціальних мереж і новинних сайтів. Він може працювати з будь-яким клієнтським середовищем і може доставляти контент практично в будь-якому форматі (включаючи HTML, RSS-канали, JSON, XML)

Django допомагає розробникам уникнути багатьох поширених помилок безпеки, надаючи платформу, яка була розроблена для «правильних дій» для автоматичного захисту сайту. Наприклад, Django надає безпечний спосіб управління обліковими записами користувачів і паролями, уникаючи поширених помилок, таких як розміщення інформації про сеанс в файли cookie, де вона вразлива (замість цього файли cookie просто містять ключ, а фактичні дані зберігаються в базі даних) або безпосереднє зберігання паролів, замість хеша пароля.

Django за замовчуванням забезпечує захист від багатьох вразливостей, включаючи впровадження SQL-коду, міжсайтовий скриптинг, підробку міжсайтових запитів і перехоплення кліків.

Django використовує засновану на компонентах (кожна частина архітектури незалежна від інших і, отже, може бути замінена). Чіткий поділ між різними частинами означає, що він може масштабуватись для збільшення трафіку шляхом додавання обладнання на будь-якому рівні: сервери кешування, сервери баз даних або сервери додатків.

Код Django написаний з використанням принципів і шаблонів проектування, які заохочують створення підтримуваного і повторно використовуваного коду. Django також сприяє гуртування пов'язаних функціональних можливостей в повторно використовувані «додатки» і, на більш низькому рівні, групує пов'язаний код в модулі

(відповідно до шаблону Model View Controller (MVC)).

Багато людей, які мало працювали з Django думають, що це просто система управління контентом. Насправді це програмний інструмент, призначений для створення і запуску веб-додатків.

Походження назви фреймворка є ключем до розуміння його багатогранної природи. Фреймворк Django зобов'язаний своїм ім'ям джазовому гітаристу Django Reinhardt, який зміг грати чудові партії на своїй гітарі, хоча два його пальці були паралізовані після аварії. Точно так і Django може виконувати безліч завдань. Django може бути використаний для створення:

- Систем управління взаємовідносинами з клієнтами (CRM);
- Системи управління контентом (CMS) для внутрішнього і комерційного використання;
- Комунікаційні платформи;
- Платформи для управління документами;

Серед іншого, Django відмінно підходить для:

- Генераторів на основі алгоритмів;
- Вирішення проблем пов’язаних з електронною поштою;
- Системи верифікації;
- Системи фільтрації з динамічно змінюваними правилами і розширеними параметрами;
- Рішення для аналізу даних і складних розрахунків;
- Машинне навчання;

Django - відмінний вибір для веб-розробки. Давайте розглянемо більше переваг цього фреймворку.

У Django неймовірно багата екосистема. Існує дуже багато сторонніх додатків, які поставляються з Django. Ці додатки можуть бути інтегровані в залежності від вимог проекту. Щоб уявити це краще, подумайте про Лего. Є багато різних блоків Lego. При розробці додатків «блок» авторизації або «блок» відправки електронної

пошти присутні майже в кожному проекті. Django складається з безлічі додатків, таких як авторизація та відправка електронних листів, які можна легко підключити до системи.

Джанго існує вже 11 років і пройшов етапи значного вдосконалення. Багато що було доведено до досконалості і багато нового було додано. Найголовніше, коли ви намагаєтеся зрозуміти, як щось має працювати в Django, ви зазвичай можете знайти відповідь. Тисячі людей, мабуть, вже вирішили будь-яку проблему, з якою ви маєте справу, і ви можете знайти рішення, надане пристрасною спільнотою Django.

Панелі адміністратора призначені для того, щоб допомогти вам керувати вашим додатком. Панель адміністратора Django створюється автоматично з коду Python, тоді як створення панелі адміністратора вручну займе багато часу і буде абсолютно безглуздим. Завдяки стороннім додаткам в адмін-панелі Django є багато можливостей для налаштування. Крім того, Django дозволяє модифікувати інтерфейс сторонніми оболонками і додавати інформаційні панелі, що відповідають вашим потребам.

Python славиться наявністю легким для читання кодом, і це є перевагою, якщо ви хочете, щоб ваш сайт займав високе місце в результатах пошуку. З Django ви можете створювати легкі для читання URL-адреси і посилання на веб-сайтах, використовуючи найбільш релевантні ключові слова та рекомендації по пошукової оптимізації (SEO).

Зрештою, доменне ім'я просто читається людиною як рядок, який зіставляється з «дружнім до комп'ютера» набором чисел, відомим як IP-адреса. Люди зациклюються на отриманні правильного доменного імені, але, як правило, нехтують тим, що такий URL - Django може це виправити.

Django є таким, що підключається за своєю дуже просто і може бути розширений за допомогою модулей. Модулі - це програмні компоненти, які дозволяють розробникам додавати в додаток певну функцію, залишаючи широкі можливості для настройки. Існують сотні пакетів, які допоможуть вам додати карти Google, створити складні дозволи або підключитися до Stripe для обробки платежів. І якщо вам потрібно масштабувати свій проект в майбутньому, ви можете відключити деякі компоненти і замінити їх іншими, які відповідають вашим поточним потребам.

Кожна мова програмування поставляється з власним набором бібліотек для вирішення спільних завдань. Бібліотека програмного забезпечення включає в себе попередньо написаний код, класи, процедури, сценарії, дані конфігурацій і багато

іншого. Як правило, бібліотека додається в програму для забезпечення більшої функціональності або автоматизації процесу без написання нового коду вручну. Це скорочує час завершення написання додатку.

Django дозволяє розробникам використовувати бібліотеки при створенні будь-якого проекту. Деякі популярні бібліотеки включають каркас Django REST, який відповідає за створення інтерфейсів прикладного програмування (API); Django CMS, яка призначена для управління контентом сайту; і Django-allauth, який являє собою інтегрований набір додатків Django для аутентифікації, реєстрації, управління своїм обліковим записом та сторонньої (соціальної) аутентифікації облікового запису.

Django цінується за своє об'єктно-реляційне відображення, яке допомагає розробникам взаємодіяти з базами даних. Об'єктно-реляційний картограф (ORM) - це бібліотека, яка автоматично передає дані, що зберігаються в базах даних, таких як PostgreSQL і MySQL, в об'єкти, які зазвичай використовуються в коді програми. На рис. 4.3.1 можна побачити як ORM забезпечує міст між реляційною базою даних та обє’ктами в Python.

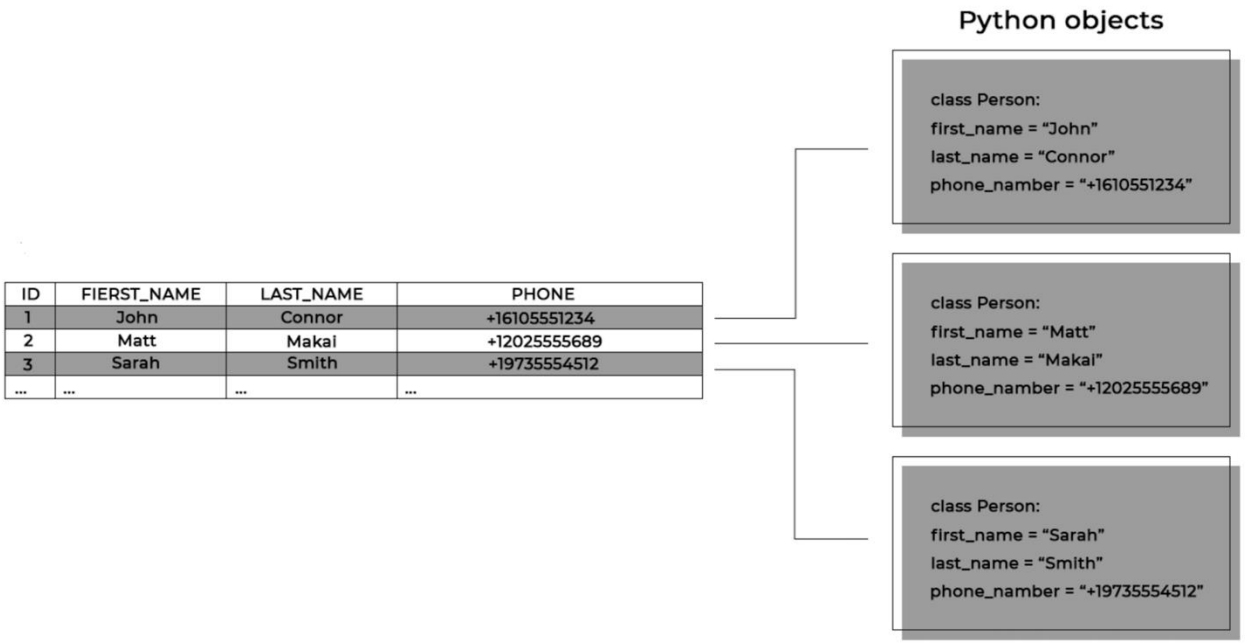

Рис. 4.3.1- Зв'язок між БД та об’єктами Python

Здатність Django ORM витягувати інформацію прискорює розробку веб-додатків і допомагає розробникам створювати робочі прототипи в найкоротші терміни. Розробникам не обов'язково знати мову, використовувану для взаємодії з базою даних для маніпулювання даними.

Крім того, ORM Django допомагає розробникам перемикатися між реляційними базами даних з мінімальними змінами коду. Це може дозволити вам використовувати SQLite для локальної розробки і, наприклад, переключитися на MySQL. Однак, як правило, найкраще використовувати одну базу даних, щоб уникнути помилок, які можуть виникнути під час переходу.

Незважаючи на велику кількість плюсів є й мінуси. Django не підходить для невеликих проектів. Django іноді може бути надмірним, але Python дозволяє використовувати інші фреймворки для розробки простих рішень. Наприклад, якщо вам потрібно створити простий чат, Django може бути занадто великою платформою, і ви можете замість цього використовувати Flask, фреймворк мікросервіса.

Також немає підтримки за замовчуванням для WebSockets. WebSockets дозволяють оновлювати інформацію або події в режимі реального часу. Django поки не підтримує веб-додатки в реальному часі. Тому вам потрібно використовувати інші фреймворки, такі як aiohttp.

Поведінку Джанго іноді складно налаштувати. Деякі внутрішні модулі Django, такі як панель адміністратора, складно налаштувати через філософію Django. Наприклад, якщо ви хочете додати посилання, динамічну статистику або щось унікальне, що не включене в екосистему Django, це може буквально зайняти весь день вашої роботи.

Отже, переваги Django переважують його недоліки і саме через те що він простий, надійний і прозорий я обрав його для своєї системи.

4.4 Docker

Docker - це набір інструментів, який спрощує створення контейнерів та контейнерних «зображень». Завдяки простому набору команд: run, build, volume і т. д.

Він надав простий для розуміння інтерфейс для створення і розгортання контейнерів. Це дозволило йому стати першим середовищем виконання, яке популяризувало контейнери і зуміло довести їх до мас. У міру розвитку він об'єднував надійне середовище для створення, розгортання та управління практично всіма компонентами життєвого циклу контейнера. Він став універсальним інструментом контейнеризації, який забезпечує фактичне середовище, задіяне у впровадженні мікросервісів.

Але в той час як його ім'я стало синонімом всіх контейнерів, Docker - тільки одна частина рішення. Він використовує простір імен Linux, можливості груп управління, профілі безпеки, мережеві інтерфейси і правила брандмауера для ізоляції процесів і забезпечення безперебійної роботи контейнера.

Docker надає інтерфейс, який дозволяє застосовувати обмеження ресурсів до контейнера. Це може використовуватися, щоб вказати, скільки пам'яті, ЦП або інших спеціалізованих ресурсів (таких як GPU) може бути використано програмою, запущеної в контейнері. Управління ресурсами дає багато переваг під час розробки: воно дозволяє імітувати спеціалізовану цільову середу (наприклад, вбудовану в пристрій) з високою точністю або гарантує, що програма не використовує ресурси особистого розробника.

Один з основних вкладів Docker в створення контейнерів полягав у тому, як він створює і зберігає зображення. Образ Docker містить середовище, залежності та конфігурацію, необхідні для запуску програми як частини файлової системи тільки для читання. Саме зображення складається з ряду шарів, де кожен шар представляє інструкцію (зазвичай подану у вигляді рядка в Dockerfile) і результати її виконання. Кожен шар в зображенні використовує стратегію копіювання при записі для збереження змін. Це дозволяє обмінюватися файлами між усіма верствами зображення і забезпечує ефективний спосіб фіксувати відмінності. Формат шару докера став стандартом для контейнерів і дає машинам швидкий спосіб обміну зображеннями між собою або між серверами зберігання, званими регістрами.

Такі сервіси, як Amazon Web Services (AWS), Google Compute Platform (GCP) і Windows Azure, охопили Docker і створили навколо нього керовані сервіси. До них відносяться керовані Kubernetes (EKS, GKE і AKS) і захищені реєстри. На відміну від інших хмарних сервісів, де вони можуть бути сильно прив'язані до конкретного

постачальника, контейнер, який працює в екземплярі Amazon Kubernetes, може бути легко перенесений для запуску в еквіваленті Google або Azure.

Переміщення зображень контейнера - проста команда. Використовуючи ідентифікатор контейнера, можна відправити зображення контейнера в будь-який реєстр, до якого у комп'ютера або користувача є доступ. Реєстри можуть бути відображені і включені в робочі процеси, пов'язані з розгортанням, безпекою або аудитом відповідності. Формат зображення включає в себе можливість пов'язувати метадані та теги версій, тому можна відстежувати зміни і використовувати систему збирання для усунення помилок.

Завдяки своїй узгодженості та мобільності контейнери Docker легко інтегруються в робочі процеси розробки. Для розробників, що створюють додаток, який вимагає підтримки компонентів (таких як база даних, чергу повідомлень або допоміжні мікросервіси), створення пов'язаних контейнерів, що надають пов'язані системи, дуже проста задача. Аналогічно, також легко створити середовище розробки в контейнері Docker, яке імітує виробничу мета і дозволяє розробникам монтувати в неї локальні вихідні файли для роботи. У стандартизованих / повторюваних середовищах розробки, складання, тестування і деплой ви можете покласти на свої контейнери, щоб робити те, що вони повинні робити кожен раз, і при цьому забезпечувати зручний доступ для розробки.

Коли розробка відбувається в контейнерному середовищі, весь життєвий цикл програми може бути виробничо-паралельним. Це означає, що з моменту виправлення проблеми до розгортання часто відбувається прямий постріл. Безперервна інтеграція і безперервне розгортання (CI / CD) - це популярна практика DevOps для автоматизації процесів збирання, тестування, підготовки і розгортання додатків. Багато конвеєрів CI / CD використовують Docker для цього.

Середовище виконання Docker включає в себе клієнтський движок (сервер), мережа, а також і інструменти для роботи з зображеннями рис. 3.4.1

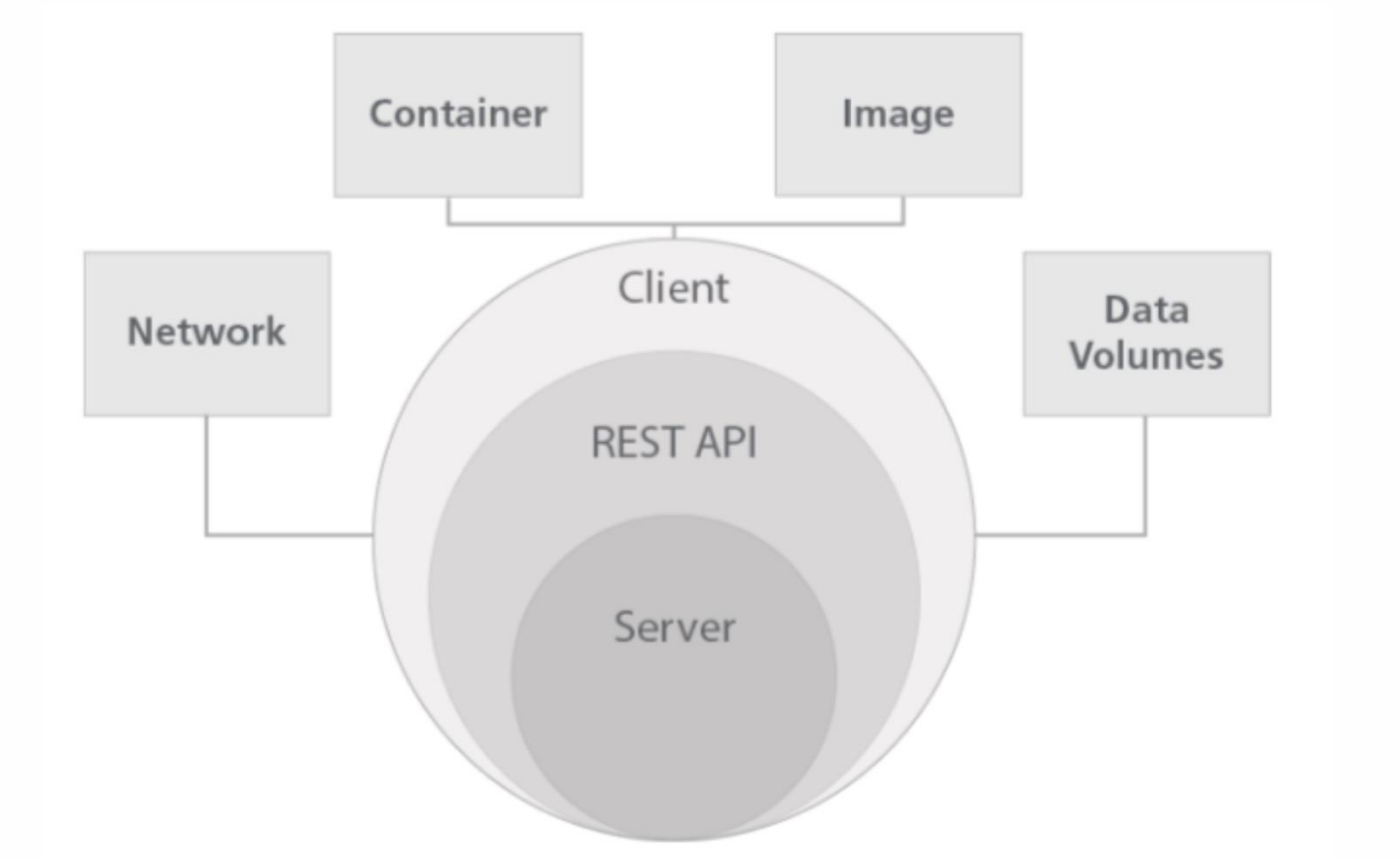

Рис. 3.4.1-Загальний огляд архітектури докера

Docker Client використовується для управління контейнерами, образами, мережами, томами і іншими ресурсами через інтерфейс командного рядка (CLI). Цей докер на основі командного рядка підключається до компонентів сервера і видає команди для API RESTful, що надається сервером. Цей клієнт є основним інтерфейсом, використовуваний більшістю користувачів.

Docker Server (розгорнутий як системний образ під назвою container) виконує адміністративну роботу по створенню і управлінню контейнерами, передачі зображень, створення мереж, налаштування томів і розподілу системних ресурсів.

Реєстр Docker - це загальнодоступний або приватний репозиторій образів контейнерів, доступ до якого можна отримати за допомогою клієнта Docker (або будь-яких засобів автоматизації, здатних зчитувати API реєстру). Docker запускає публічний реєстр Docker Hub. Docker Hub включає в себе тисячі загальнодоступних образів, що містять операційні системи, утиліти, бази даних і сервери додатків.

Контейнери надають безліч привабливих функцій для команд розробників. Вони включають в себе одиницю упаковки, одиницю розгортання, одиницю повторного використання, одиницю виділення ресурсів і одиницю масштабу. По суті, він забезпечує ідеальний набір інструментів для розробки і розгортання мікросервісів.

Контейнери також представляють велику цінність для робочих груп. Оскільки вони можуть забезпечити багато переваг віртуалізації, але без накладних витрат, вони надають можливість значно поліпшити процеси. Однак у багатьох важливих аспектах вони також змінили уявлення про те, як слід розробляти і розгортати програмне забезпечення і використовувати нові обчислювальні платформи.

DevOps - це комбінація інструментів, практик і принципів, які підвищують здатність організації надавати додатки і послуги з високою швидкістю. DevOps, як практикується в більшості компаній-розробників програмного забезпечення, включає в себе контейнери. Контейнери дозволили «хмарним» обчисленням розвиватися в напрямку більшої абстракції. Повністю абстрагована хмарна платформа видаляє всі деталі того, як програма упаковується і надається. З точки зору розробника, вона просто виконується.

Інфраструктура як код - це практика використання тих же інструментів і методів, які використовуються для розробки програмного забезпечення в управлінні інфраструктурою і обчислювальними ресурсами. На практичному рівні це означає, що конфігурація додатку буде зберігатися в системі контролю версій, проаналізованою і протестованою.

Контейнери забезпечують конкретну реалізацію інфраструктури як коду. На самому базовому рівні Dockerfiles створюють середовища виконання додатків, які потім можуть бути об'єднані за допомогою маніфестів і розгорнуті в зв'язкові групи ресурсів. У більш складних сценаріях контейнерні системи можуть запитувати спеціальні ресурси, такі як графічні процесори і спеціальні форми зберігання.

Безперервна інтеграція / безперервне розгортання (CI / CD). CI / CD - це практика розробки автоматизованих пакетів, які можуть тестувати, перевіряти, ставити і розгортати код. Ефективний конвеєр CI / CD дозволяє доставляти оновлення програм з дуже високою частотою. Такі компанії, як Netflx, Amazon, Etsy і Google, часто можуть оновлювати розгорнуту систему десятки разів в день.

Існує багата екосистема інструментів, призначених для включення CI / CD на основі контейнерів. Наприклад, Jenkins - це інструмент автоматизації CI, який може автоматично створювати контейнери з вихідного коду, запускати на них тестові набори для забезпечення їх функціональності і розгортати отримані контейнери в реєстрі, якщо вони пройдуть тести. Spinnaker - це інструмент, здатний відстежувати реєстр для оновлених збірок контейнерів і розгортати їх в проміжному або виробничому кластері.

Контейнери здатні забезпечити безсерверну функціональність. Такі системи, зазвичай звані «безсерверними», дозволяють розробнику написати простий фрагмент коду та розгорнути його на функціональній платформі, яка потім виконує завдання з розгортання, тестування і виконання.

Отже, використовуючи Docker для розробки, керованої контейнером, узгодженість, переносимість та ізоляція, що забезпечується контейнерами, забезпечують більш швидку реалізацію програмного забезпечення. Контейнери є надзвичайно потужним інструментом розробки, здатним забезпечити величезну цінність для окремих команд і цілих організацій.

4.5 Backbone.js

Backbone - це інфраструктура JavaScript, яка орієнтована на структурний код веб-додатку шляхом створення моделі MVC і підключення її до існуючого API через інтерфейс RESTful JSON. Розглянемо кілька причин вибрати саме цю бібліотеку:

- Моделі привязані до ключ-значення і призначені для користувача події.
- Відображення з декларативною обробкою подій.
- Колекції з багатим API перелічуваних функцій.

Backbone моделі є ядром усіх додатків JavaScript і включають в себе

внутрішню таблицю атрибутів даних і ініціюють події, якщо там виявляються які-небудь зміни. Вони також доповнені інтерактивними даними і величезною частиною логіки коду навколо них: перетворення, період дії, властивості комп'ютера, контроль доступу.

Backbone View призначені для візуалізації необхідних даних з моделей. Однак вони також можуть бути незалежними порожніми планшетами призначеними для користувача інтерфейсу без даних. Тим часом моделі повинні бути зазвичай пов'язані з виглядами. Крім того, вони повинні реагувати на пов'язані зі зміною моделі події, а також реагувати і відображати себе відповідно до завдань моделі.

Основна ідея виглядів по логічним структурам - інтерфейс програми. Більш того, вони підкріплені моделями, які можуть оновлюватися окремо, якщо відбуваються якісь зміни, без необхідності переробки всієї сторінки в цілому. Це позбавляє розробника від пошуку об'єктів JSON в коді програми, коли необхідно знайти елементи DOM. Отже, вигляд просто показує всі закріплені моделі і де вони відображаються в інтерфейсі.

Backbone Collection складні з групами моделей і допомагають легко управляти ними. Колекції надають розробникам веб-додатків контроль над завантаженням та збереженням нових моделей на сервер і спрощують агрегацію або копіювання продуктивності в списках моделей. Вони дозволяють розробнику відслідковувати:

- Подію змінення, в той час як одна з моделей була змінена.
- Подію додавання і видалення, витягують цілі колекції моделей з боку сервера.
- Спостерігати за будь-якою конкретною зміною обраних моделей або цілих колекцій.
- Більш того, незважаючи на стандартні події збору, на кожну з них можуть вплинути будь-які модельні події.

Події - це мінімальна, але надійна реалізація шаблону проектування спостерігача для об'єкта JavaScript. Вони з'єднують веб-додаток разом і роблять його інтерактивним. Таким чином, кожен з об'єктів Backbone містить систему подій і дозволяє їм ініціювати події і обробляти їх. Зрештою, всі події можуть бути розділені на ті які:

- Вмикаються коли відбуваються події.
- Вмикаються коли дія завершилась.
- Реагують на тригер поставлений на певну подію.

Backbone Router надає параметри для маршрутизації клієнтської частини веб-додатку і управління ним за допомогою подій. Кількома роками раніше ця функція створювала тільки хеш-фрагменти, такі як #page. Але тепер, коли API Backbone History підтримується багатьма браузерами, всі типи URL можуть легко використовуватися в їх стандартних формах. Всі посидання маршрутизуются одним API.

Backbone History надає розробникам функцію відстеження історії і відповідає відповідним маршрутом. Він запускає зворотні виклики для управління подіями і включає маршрутизацію в веб-додатку. В цьому випадку розробникам не потрібно створювати будь-які спеціальні події, тому що історія вже має їх:

- Події Hashchange.
- PushState.
- Callbacks.

Backbone Sync - це основна функція Backbone, яка дозволяє перевіряти і зберігати модель на сайті сервера веб-додатку. Зазвичай він використовує дію ajax для того, щоб зробити запит RESTful JSON для повернення jqXHR. З іншого боку, розробник може перевизначити його, щоб використовувати інший належний додаток для стратегій проекту:

- XML
- Локальна пам'ять.
- WebSocket.

У той час як нам потрібно розробити додаток, Backbone виявиться дуже корисний. Каркас Model View допоможе набагато більше, ніж просто структурування архітектури Javascript. Отже, Backbone вирішить безліч проблем, з якими ми можемо зіткнутися в кінці розробки нашої програми. Зазвичай більшість даних веб-додатку містять серверну частину. В результаті HTML підштовхується до повної перезавантаження сторінки, в той час як він приймає будь-яку дію призначеного для користувача інтерфейсу. Тим часом, клієнтський JavaScript має обмежені рішення для вирішення цієї проблеми. Але з Backbone.Js все змінилося. Він завантажує необроблені дані з серверної частини веб-додатку і відображає їх в браузері відповідно до необхідності.

Backbone використовує одне подання для управління декількома вкладеними представленнями, які спільно використовують необхідні моделі. Це допомагає заощадити час, підвищити читаність коду і спростити будь-яке перекодування в майбутньому. В результаті, це відбувається, коли розробники намагаються створювати складні проекти з великим інтерактивним і багатофункціональним призначеним для користувача інтерфейсом.

Традиційно Backbone підштовхує розробників до поділу шарів даних і представлень в загальних поняттях. Тим часом, більшість розробників рекомендує робити це у всіх випадках, за винятком моделей, які не створюються з посиланнями на їхні уявлення. Отже, всі моделі і колекції, всі верстви даних в кінці повинні бути чітко відокремлені від уявлень, до яких вони належать. Суворий поділ концепцій допоможе уникнути перетворення всієї роботи розробника в непрацюючий код. Тому це допоможе в навігації по коду і заощадить час, якщо в майбутньому буде потрібно виправити будь-які помилки.

Таким чином, розроблений додаток буде завантажувати тільки одне корисне навантаження, яке включає в себе всі необхідні сценарії, стилі і мітки. Все, що потрібно користувачеві для здійснення простих дій, буде в одному місці. В результаті немає ніяких вимог для будь-якої додаткової фонової діяльності. Наприклад, Gmail легко перемикається між читанням ваших листів про доходи і відповіддю на них, тому що там немає ніякого додаткового запиту до сервера.

Вага проекту дуже важлива, коли ми говоримо про швидкість завантаження і

адаптивність мобільного додатку. Backbone - це перспективна структура у всіх областях веб-розробки:

- Розмір магістралі становить близько 7,6 КБ, в той час як він мінімізований і стиснутий.
- Можна легко позбутися залежності від jQuery.
- Backbone заснований на бібліотеках Underscores.Js (5.7kb), яка може стати в нагоді при розробці.

Можливість розширення - головна ідея всієї основи Backbone. Таким чином, він включає в себе багато невеликих бібліотек, які підходять для спеціалізованих потреб. Більш того, завжди є можливість створити свій власний фреймворк MVC:

- LayoutManager - забезпечує рендеринг виду.
- Backbone.Stickit і Epoxy - забезпечує кращу прив'язку виду до моделі.
- Marionette - дозволяє краще структурувати код програми.

З тих пір як версія 0.1.0 була випущена 5 років тому, цей час її розробка і оновлення ніколи не припинялися. Тим не менш, це може бути легко вивчено через багато речей:

- документація та посібники доступні безкоштовно в Інтернеті.
- творець Backbone опублікував докладний посібник для цієї платформи.
- Backbone був добре перевірений і підтриманий різними додатками.

Використання Backbone дозволяє легше структурувати JavaScript. Він також включає в себе основи об'єктно-орієнтованого програмування. В результаті він переходить від використання DOM до RESTful API для отримання JSON і збереження всіх даних в форматі моделей. Нарешті, зв'язок між поданням і моделями дозволяє побачити будь-які зміни HTML.

Абстрагування - одна з трьох речей, які дійсно мають значення в програмуванні. Чим воно краще, тим краще буде код. Він приховує всі дані, крім відповідних даних про об'єкт де необхідно спростити навігацію в складних проектах і підвищити ефективність.

Простіше кажучи, Backbone дозволяє розробнику відокремлювати логіку додатка від призначеного для користувача інтерфейсу. В результаті обидва вони можуть бути легко змінені, оновлені і підтримані при необхідності. Завдяки цьому поділу стає більш зрозумілим, де потрібні зміни і який модульний тест повинен бути написаний.

5 Робота користувача з програмним продуктом

В цьому розділі приведені системні вимоги для роботи з додатком та сценарії роботи користувача з ним.

5.1 Системні вимоги

Для забезпечення коректної та безвідмовної роботи інформаційної системи ефективного моніторингу хронічних захворювань персональний комп’ютер повинен мати процесор не гірше, ніж Intel ® Pentium ® / Celeron ® / Xeon™ або з тактовою частотою не менше 1,8 GHz або AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ для користувачів процесорів від фірми AMD. Також комп’ютеру користувача повинно бути доступно не менше 4 Gb оперативної пам’яті та графічне ядро не гріше, ніж Intel ® HD Graphics, що еквівалентно графічним картам з об’ємом пам’яті не менше, ніж 1 GB.

5.2 Користувацький інтерфейс

Головне меню адміністративної панелі Wagtail має наступний вигляд (Рисунок 5.2.1):

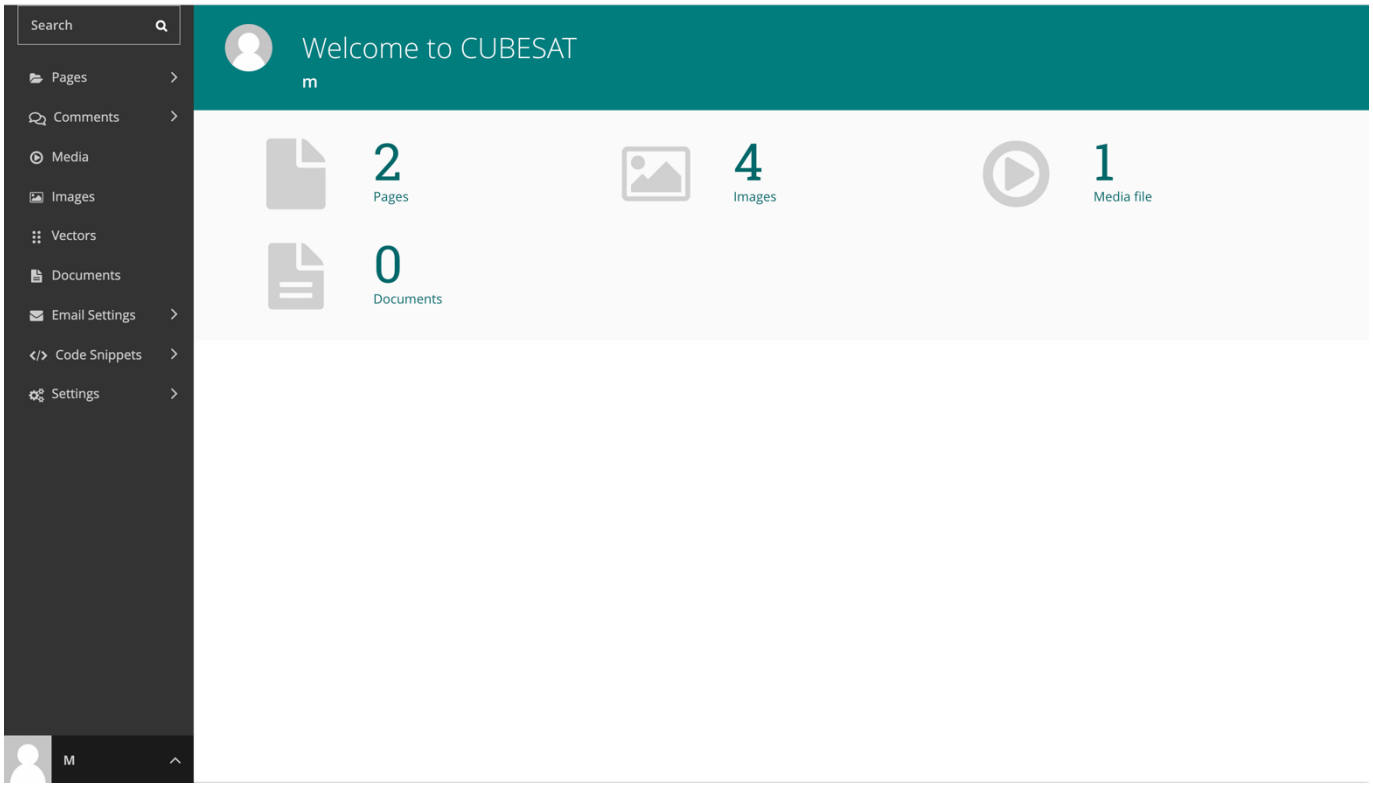

Рисунок 5.2.1 – Головне меню

Для початку роботи користувач повинен створити головну сторінку сайту. Для цього потрібно налаштувати її у вкладці “Pages”(рисунок 5.2.2).

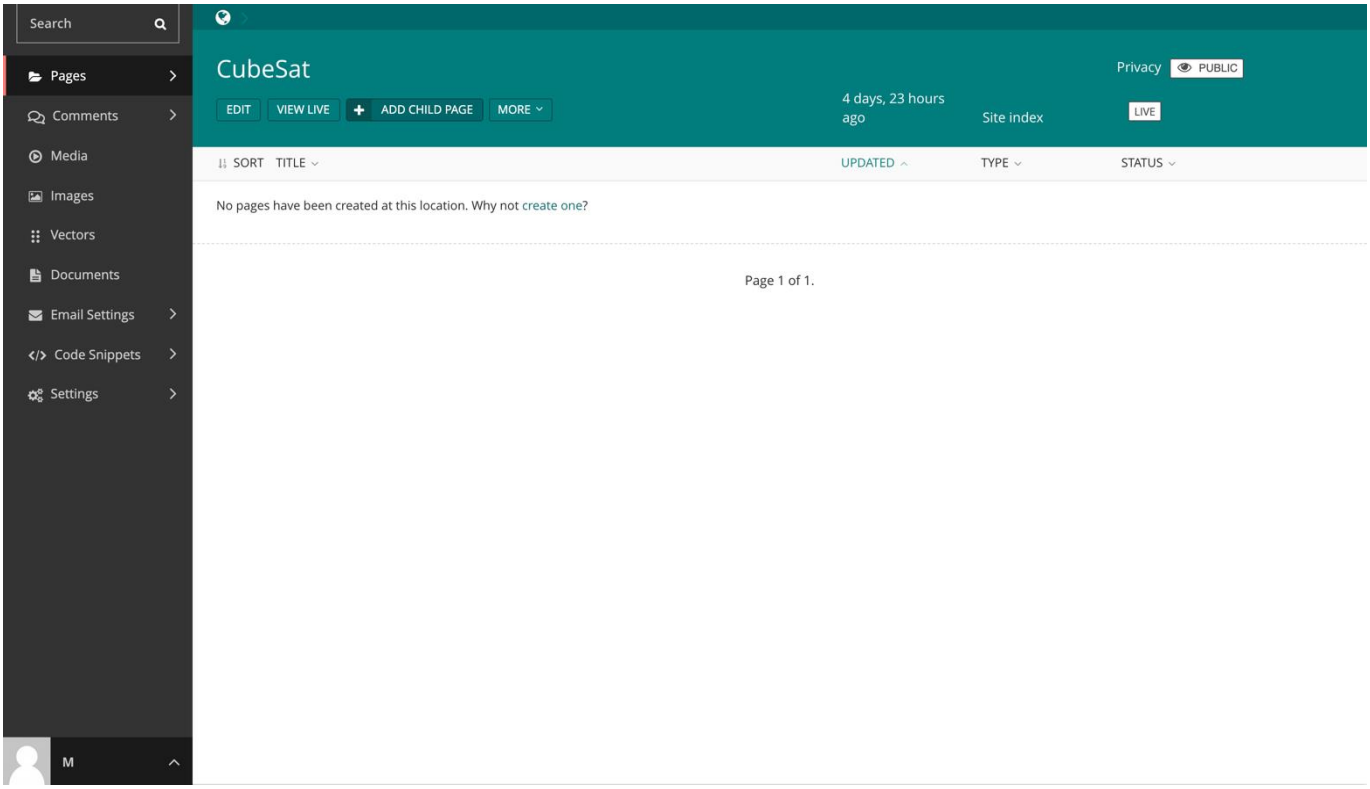

Рисунок 5.2.2 – Створення головної сторінки сайту

Після створення головної сторінки потрібно наповнити її контентом. Для цього вже створені блоки. Система дає можливість сворити поля Streamfield, послідовність "блоків" вмісту, яку можна переставити за потреби. Блоки дозволяють створити багато різноманітного контенту, що в подальшому опублікується на сайті(Рисунок 5.3):

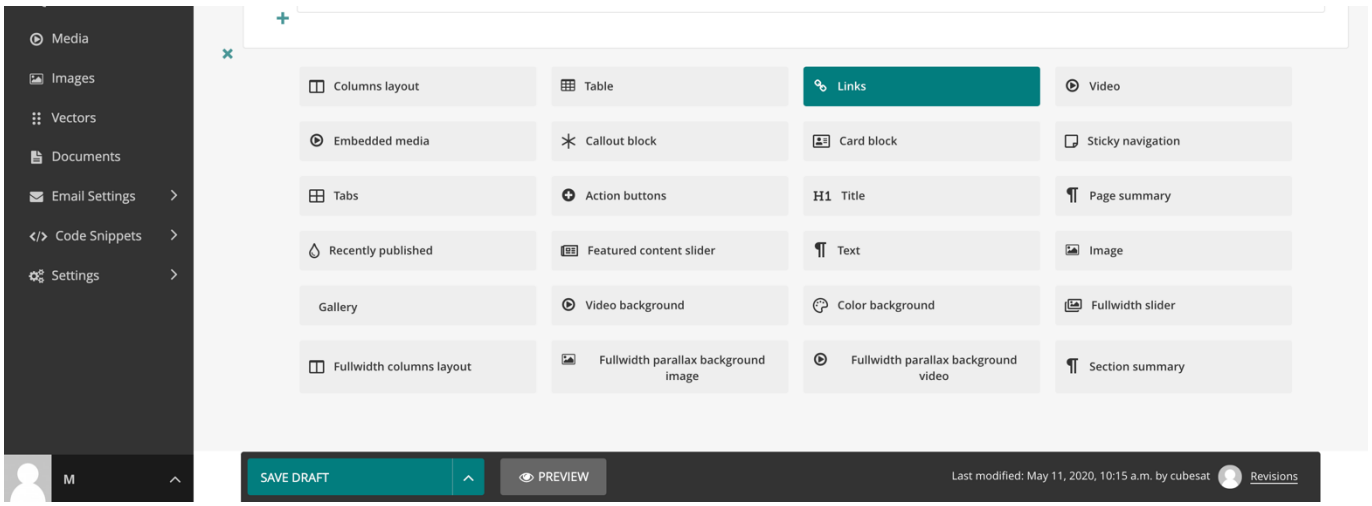

Рисунок 5.2.3 – Блоки контенту

Після створення головної сторінки її потрібно опублікувати на сайті. Для цього потрібно натиснути кнопку в лівому нижньому кутку “Publish” (Рисунок 5.2.4):

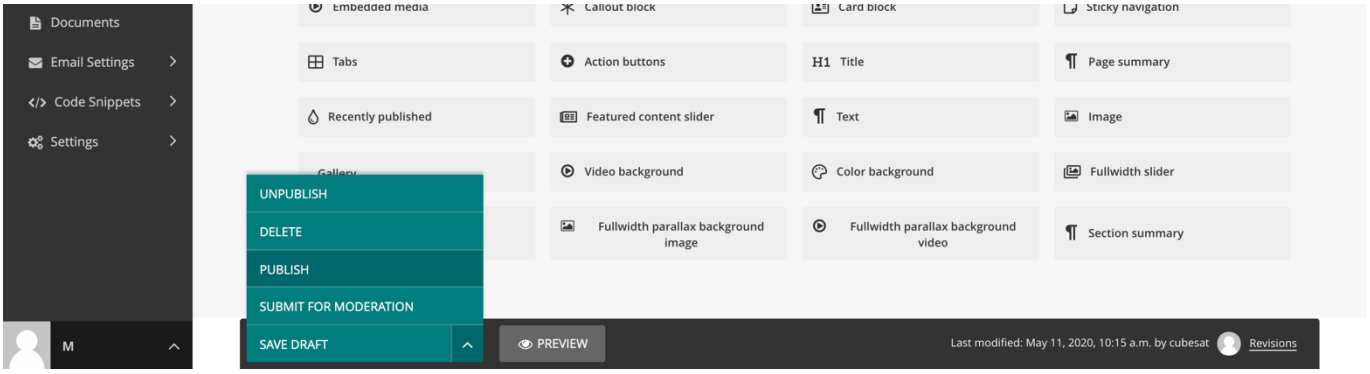

Рисунок 5.2.4 – Публікація сторінки

Після публікації, сторінка буде доступна для перегляду користувачам. Також є можливість запланованої публікації, тобто сторінка (наприклад нова стаття про новини космосу) опублікується у вказаний час(Рисунок 5.2.5):

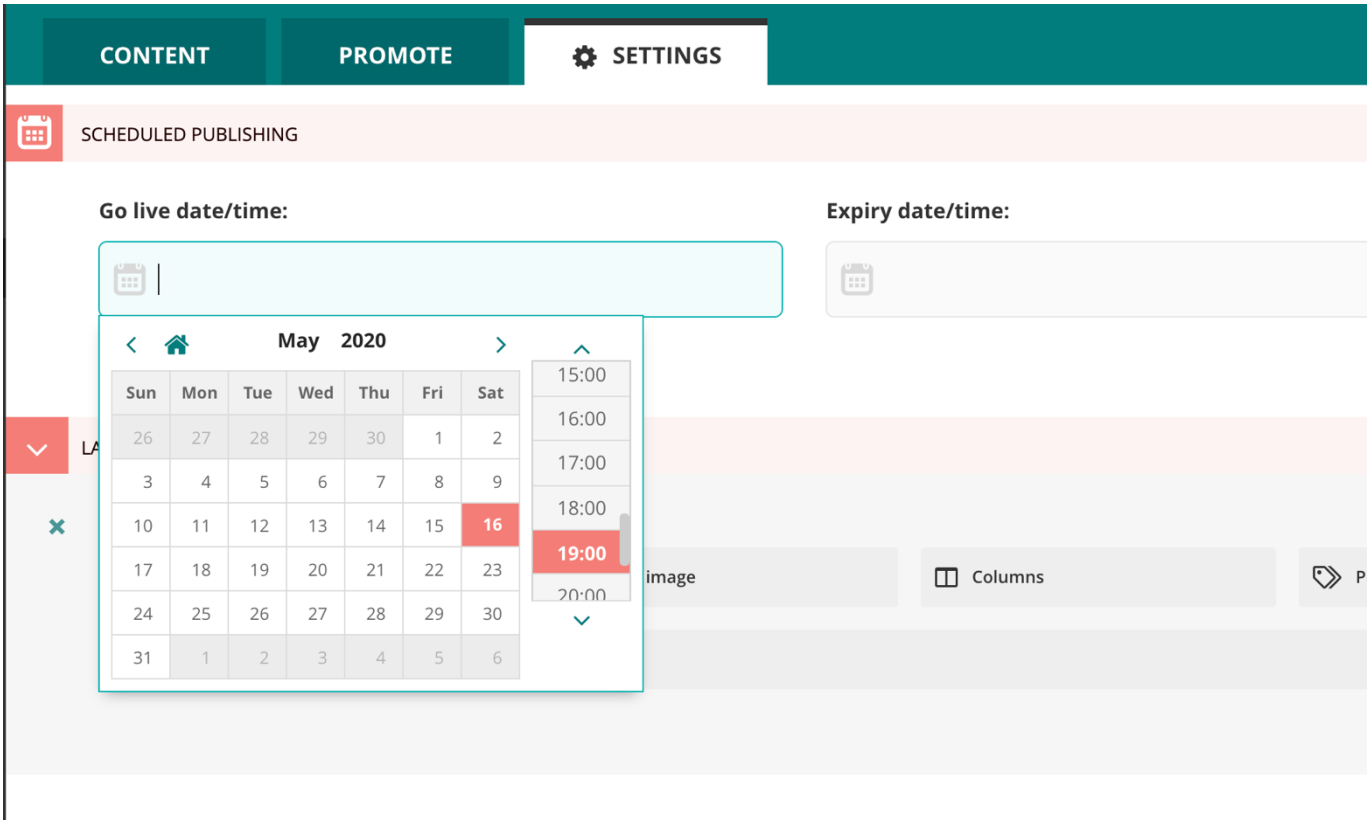

Рисунок 5.2.5 – Запланована публікація

При розробці системи моніторингу був врахований той факт, що сайт постійно ростиме, та наповнюватимиться контентом в якому з часом буде складно орієнтуватися. Тому було вирішено зробити пошук. Він дає змогу швидко знайти потрібний пост в блозі, картинку, відео чи інший контент на сайті. Для пошуку користувач повинен скористатися пошуковим полем у верхній лівій частині адміністративної панелі та натиснути кнопку «Enter». (Рисунок 5.2.6):

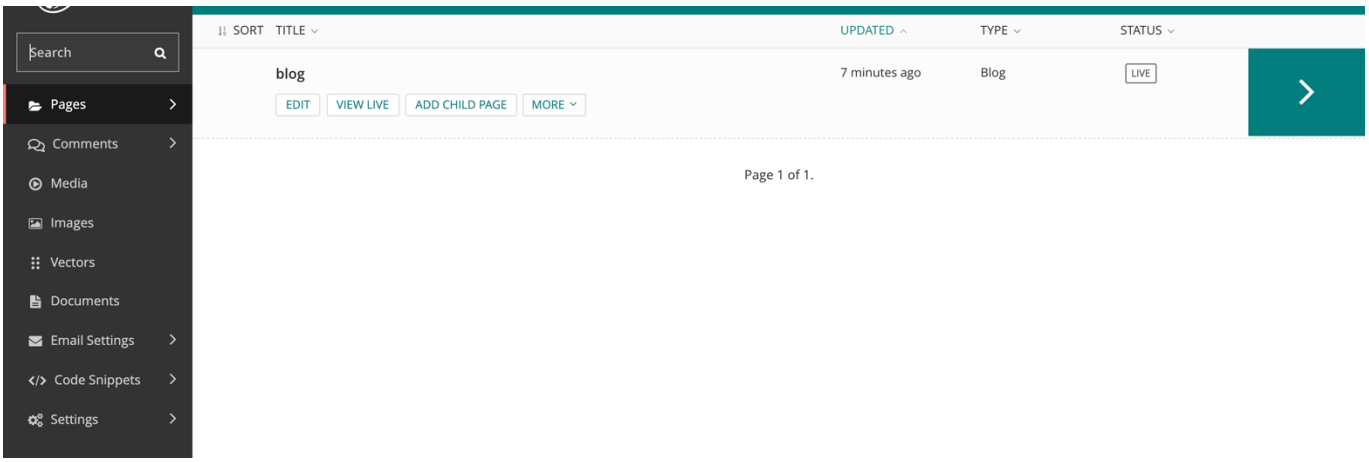

Рисунок 5.2.6 – Пошуковий рядок

ВИСНОВКИ

У ході аналізу існуючого програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників було досліджено системи, які слугують для вирішення поставлених задач. Аналіз показав, що існуючі системи вирішують задачу не у повному обсязі та є досить малими.

Розроблений програмний продукт є актуальним та буде актуальним в подальшому, оскільки розробляється як частина великої системи, яку з часом можна розширити. Розроблена система дозволяє легко інтегрувати додаткове обладнання, та його налаштувати.

Програму можна використовувати для моніторингу наносуптників та розвитку сфери їх побудови і запуску шляхом наповнення сайту контентом.

В ході роботи було проведено огляд та зроблено аналіз засобів, що були використані для створення даного програмного забезпечення (середовища розробки Intellij IDEA PyCharm, Django Framework та засобів створення веб-інтерфейсів: HTML5, CSS, JavaScript та Backbone.js).

Для роботи з даним програмним забезпеченням необхідний лише комп’ютер середньої потужності.

Користувач має змогу не тільки спостерігати за наносупутниками але й власноруч наповнювати сайт різною цікавою інформацією.

За результатами виконання тестів підтверджена коректність роботи програми, отже система відповідає поставленим вимогам.

Користувачами системи можуть бути будь-які користувачі, які здійснюють нагляд за системою.

Робота над дипломним проектом покращила знання різноманітних технологій, що використовуються під час розробки програмного забезпечення. Також було створено декілька прототипів програмного забезпечення, які вирішували різноманітні аспекти поставленої задачі, а також які лягли в основу розробленого програмного забезпечення.

Систему було написано на мові програмування Python з використанням фреймворку Django. Графічний веб-інтерфейс був написаний за допомогою бібліотеки Backbone.js, JavaScript, HTML5 та SCSS .

Створений програмний продукт дозволяє користувачу не тільки спостерігати за наносупутниками але й власноруч наповнювати сайт різною цікавою інформацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robert C. Martin— Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition / Robert C. Martin, 2008. — 413 с.
2. Stoyan Stefanov— JavaScript Patterns / Stoyan Stefanov, 2010. — 218 с.
3. Mark Lutz — Learning Python, 3rd Edition / Mark Lutz, 2007. — 832 с.
4. Martin Gruber — Understanding SQL / Martin Gruber, 2001. — 291 с.
5. Jet Brains // IntelliJ IDEA overview [Електронний ресурс] — Режим доступу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html
6. Hosting Review // Website Interface [Електронний ресурс] — Режим доступу: https://hosting.review/tutorial/website-interface/
7. Techtarget Network // App Architecture [Електронний ресурс] — Режим доступу: https://searchapparchitecture.techtarget.com/definition/user-interface-UI
8. MDN Web Docks // Django introduction [Електронний ресурс] — Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
9. SQLite Tutorial // What Is SQLite [Електронний ресурс] — Режим доступу: https://www.sqlitetutorial.net/what-is-sqlite/
10. Steel Kiwi // Why Django is the Best Web Framework [Електронний ресурс] — Режим доступу: https://steelkiwi.com/blog/why-django-best-web-framework-your-project/
11. ThinkMobiles // Why use Backbone.js [Електронний ресурс] — Режим доступу: https://thinkmobiles.com/blog/why-use-backbonejs/

ДОДАТОК 1

Модернізація програмного забезпечення системи моніторингу технічного стану та режиму функціонування наносупутників

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ61122_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТМ61122_20Б 81-1	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТМ61122_20Б 12-1		Текс програмного продукту
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТМ61122_20Б 12-2		Опис програмного модулю

ДОДАТОК 2

Розробка системи моніторингу технічного стану та режиму функціонування наносупутників

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТМ61122_20Б

Аркушів 13

Київ 2020

Основні функції додатку:

```
from django.shortcuts import reverse
from django.contrib.auth.models import Permission

from wagtail.contrib.modeladmin.options import ModelAdmin as WagtailModelAdmin, \
  ModelAdminGroup as WagtailModelAdminGroup
from wagtail.contrib.modeladmin.views import CreateView as WagtailCreateView, \
  EditView as WagtailEditView
from wagtail.contrib.modeladmin.helpers import ButtonHelper, PermissionHelper
from wagtail.contrib.modeladmin.views import IndexView

from guru.errors import ConfigurationError
from guru.helpers import str2bool


class WagtailCollectionViewMixin(object):
  '''    Wagtail collection view mixin which adds a `user` argument to the form
    keyword arguments, allowing forms to filter available Collections
    based on those to which the user has access to.
  '''
  def get_form_kwargs(self, *args, **kwargs):
    '''    Fetches the form keyword arguments: in addition to parent view keyword
      arguments, adds a user instance (based on the request user). This allows
      for the view to be used to Collection member base forms.
    '''
    form_kwargs = super(WagtailCollectionViewMixin, self).get_form_kwargs(*args, **kwargs)
    if not form_kwargs.get('user') and hasattr(self.request, 'user'):
      form_kwargs['user'] = self.request.user

    return form_kwargs


class WagtailCollectionCreateView(WagtailCollectionViewMixin, WagtailCreateView):
  '''    Wagtail create view which can be used with Collection member base forms
  '''
  pass


class WagtailCollectionEditView(WagtailCollectionViewMixin, WagtailEditView):
  '''    Wagtail edit view which can be used with Collection member base forms
  '''
  pass


class WagtailCollectionPermissionHelper(PermissionHelper):
  '''    Permission helper which sets the permissions for site and email messages
    associated with registration and email validation.
  '''
  permission_policy = None

  def __init__(self, *args, **kwargs):
    super(WagtailCollectionPermissionHelper, self).__init__(*args, **kwargs)

  def _check_required_properties(self):
    '''    Ensure that the permission helper has all required properties needed
      to check Collection permissions for a specific model.
    '''
    # Model permission policy
    if not getattr(self, 'permission_policy', None):
      raise ConfigurationError('Unable to initialize Wagtail collection permission helper, '
        + 'no permission policy associated with the model instance.')

  def user_can_list(self, user):
    self._check_required_properties()
    return self.permission_policy.user_has_any_permission(user, ('add', 'change'))

  def user_can_create(self, user):
    self._check_required_properties()
    return self.permission_policy.user_has_permission(user, 'add')

  def user_can_inspect_obj(self, user, obj):
    return self.inspect_view_enabled and self.user_can_list(user)

  def user_can_edit_obj(self, user, obj):
    if callable(getattr(obj, 'is_editable_by_user')):
      return obj.is_editable_by_user(user)

    return self.permission_policy.user_has_permission_for_instance(user, 'change', obj)
```

```
  def user_can_delete_obj(self, user, obj):
    return self.permission_policy.user_has_permission_for_instance(user, 'delete', obj)


class WagtailCollectionModelAdmin(WagtailModelAdmin):

  permission_helper_class = WagtailCollectionPermissionHelper
  permission_policy = None
  base_form_class = None

  create_view_class = WagtailCollectionCreateView
  edit_view_class = WagtailCollectionEditView

  def __init__(self, *args, **kwargs):

    if not self.permission_helper_class.permission_policy and not self.permission_policy:
      raise ConfigurationError('Unable to initialize Wagtail Collection model admin. '
        + 'No permission policy associated with the model class or the model admin.')

    super(WagtailCollectionModelAdmin, self).__init__(*args, **kwargs)

    # If a base form class if provided to the admin but not the model, apply to the model
    if self.base_form_class and not getattr(self.model, 'base_form_class', None):
      setattr(self.model, 'base_form_class', self.base_form_class)

    # Check permission helper class for permission policy, if one not provided use the permission
    # policy from the model admin.
    if getattr(self.permission_helper, 'permission_policy', None) is None:
      setattr(self.permission_helper, 'permission_policy', self.permission_policy)

  def get_permissions_for_registration(self):
    return Permission.objects.none()

  def get_queryset(self, request, *args, **kwargs):
    '''    Filter objects for only those instances to which the user has access
    '''
    qs = super(WagtailCollectionModelAdmin, self).get_queryset(request, *args, **kwargs)
    return qs.filter(
      collection__in=self.permission_helper.permission_policy.collections_user_has_permission_for(request.user, 'add'))


INCLUDE_PROTECTED_QUERY_PARAM = 'include-protected'


class ProtectedContentIndexView(IndexView):
  '''    Index view that includes additional properties for showing registration
    content in Compass.
  '''
  IGNORED_PARAMS = IndexView.IGNORED_PARAMS + (INCLUDE_PROTECTED_QUERY_PARAM,)


class ProtectedContentAdmin(WagtailCollectionModelAdmin):
  '''    Wagtail model admin for managing Compass registration content:
    site messages, registration emails, validation emails, etc.
  '''
  protected_model_param = 'protected'
  index_view_class = ProtectedContentIndexView
  inspect_view_enabled = True
  inspect_view_fields_exclude = ('protected',)

  def get_queryset(self, request, *args, **kwargs):
    qs = super(ProtectedContentAdmin, self).get_queryset(request, *args, **kwargs)

    # Remove protected entries from the results (unless query flag is set to True)
    if self.model._meta.get_field(self.protected_model_param):

      # Allow admin users to view/edit protected entries by adding query flag
      if request.user.is_superuser \
        and str2bool(request.GET.get(INCLUDE_PROTECTED_QUERY_PARAM, False)):
        qs = qs
      else: qs = qs.filter(**{ self.protected_model_param: False })

    return qs
```

```
import logging

from django.shortcuts import redirect, resolve_url
from django.urls import reverse
from django.utils.http import is_safe_url
from django.views.generic.base import TemplateView

from django.contrib import auth
from django.contrib.auth.views import PasswordResetView, PasswordResetDoneView

from guru.helpers import gsetting, create_token
from guru.helpers.compatability import guru_permission_denied, guru_page_not_found
from guru.helpers.user import user_displayname
from guru.helpers.utils.object import pick
from guru.errors import ConfigurationError
from guru.views import GuruApiRestView

from wgtauth.social.views import OpenIDLoginRedirectAbstractView, \
        OpenIDLoginCallbackAbstractView
from wgtauth.registration.views import RegistrationView, RegistrationSuccessView, ConfirmEmailView, \
        SESSION_NEW_REGISTRATION_ATTR

from ..models.pages import SiteIndex
from ..models.users import UserProfile
from ..models.settings import SiteBranding
from ..forms.users import UserProfileForm
from ..views import ApiObjectMixin, ApiViewFormMixin, RootPageMixin

from .models import SocialAuthorizationServer, SocialUserAccount
from .forms import PasswordRegistrationForm

logger = logging.getLogger(__name__)


OPENID_AUTH_TOKEN_SESSION_PROVIDER_PARAM = 'openid-auth-provider'
OPENID_AUTH_TOKEN_SESSION_PARAM = 'openid-auth-token'
OPENID_AUTH_TOKEN_TYPE_SESSION_PARAM = 'openid-auth-token-type'
OPENID_AUTH_TOKEN_SCOPE_SESSION_PARAM = 'openid-auth-scope'


class OpenIDViewPropertiesMixin(object):
        '''     View mixin which implements the interface required by OpenID authentication views
        '''
        authserver_objectid_fieldname = 'pk'
        authserver_objectid_url_param = 'serverid'

        def get_auth_server(self, request, vargs, vkwargs):
                '''     Retrieve the authorization server. Caches a copy in the view keyword arguments,
                        to avoid multiple queries to the database.
                '''
                # Attempt to retrieve auth server from cache
                authserver = vkwargs.get('authserver')

                # Not available from cache, retrieve from the database and place in view keyword arguments
                if not authserver:
                        authserver = SocialAuthorizationServer.objects.prefetch_related('provider') \
                                .get(**{self.authserver_objectid_fieldname : vkwargs.get(self.authserver_objectid_url_param) })
                        vkwargs['authserver'] = authserver

                return authserver

        def application_redirect_url(self, request, vargs, vkwargs):
                '''     Retrieve the login redirect/callback URL which should be used by the authentication service
                '''
                authserver = self.get_auth_server(request, vargs, vkwargs)
                return authserver.compass_url_callback


class OpenIDLoginRedirectView(OpenIDViewPropertiesMixin, OpenIDLoginRedirectAbstractView)               :
        '''     Redirect user login requests to the specified OpenID provider for authentication. First step
        in the oAuth/OpenID authentication workflow.
        '''
        pass


class OpenIDLoginCallbackView(OpenIDViewPropertiesMixin, OpenIDLoginCallbackAbstractView):
        '''     Callback endpoint to which  should redirect users which have been authenticated
        successfully. Completes the oAuth authentication workflow.
        '''
```

```
def get_openid_username(self, authtoken, request, vargs, vkwargs):
        '''     Retrieve the OpenID username via the authorization token
        '''
        openid_username = getattr(getattr(authtoken, 'user', None), 'username', None)
        if not openid_username:
                raise ConfigurationError(
                        'Unable to retrieve OpenID username using auth token %s' % authtoken.access_token)

        return openid_username

    def get_django_user(self, openid_username, authtoken, request, vargs, vkwargs):
        '''     Retrieve the social auth profile associated with the OpenID username, and
                from that, fetch the user model.
        '''
        authserver = self.get_auth_server(request, vargs, vkwargs)

        # Attempt to retrieve social user profile via the unique provider ID (openid_username)
        try: socialuser_profile = authserver.social_user_profiles.get(social_user_id=openid_username)
        except SocialUserAccount.DoesNotExist:

                # Determine if there is currently a user logged in (link social profile to existing accout)
                if request.user.is_authenticated:
                        django_user = request.user
                        created = False

                # User not yet logged in: create a new account from the user data
                else:

                        # Construct a username for the account:
                        # 1. If available, parse the user handle portion of the email and use that
                        # 2. For accounts without email data, convert the first name and last name to lowercase
                        #    and then append them with a dot.
                        # 3. For accounts without first name and last name, use the full name and replace
                        #    spaces with a dot.
                        # 4. For accounts without any identifiers, generate a random string to use as the username
                        django_username = \
                                authtoken.user.django_username if hasattr(authtoken.user, 'django_username') \
                                else authtoken.user.email.split('@')[0] \
                                        if hasattr(authtoken.user, 'email') and '@' in authtoken.user.email \
                                else '.'.join((authtoken.user.first_name.lower(), authtoken.user.la
```

Основні блоки:

```
class HomeListingStatusHistory(GuruTokenModel):

    '''     Home listing status taken from the MLS. Keys/values maintain the schema of

            the MLS from which they were taken.



            status - status_date DatetimeField, status CharField

    '''

    listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='mls_history_status')

    status_date = models.DateTimeField(help_text='Date/Time the listing status was modified in the MLS')

    status = models.CharField(max_length=22, help_text='Current Status of the listing')


    class Meta:

            app_label = 'listings'

            verbose_name = 'MLS Status History Entry'

            verbose_name_plural = 'MLS Status History for a Home Listing'

            unique_together = ('listing', 'status_date')


    @property
```

```
def json(self):
        '''     Representation of the model that can be safely serialized to JSON
        '''
        return {
                'listing': self.listing.pk,
                'status_date': self.status_date,
                'status': self.status,
        }


class HomeListingPriceHistory(GuruTokenModel):
        '''     Home listing price taken from the MLS. Keys/values maintain the schema of
                the MLS from which they were taken.

                price - price_change_date DatetimeField, list_price IntField
        '''
        listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='mls_history_price')
        date_price_change = models.DateTimeField(help_text='Date/Time the listing price was changed in the MLS')
        list_price = models.IntegerField(help_text='Listing Price of the home listing')

        class Meta:
                app_label = 'listings'
                verbose_name = 'MLS Price History Entry'
                verbose_name_plural = 'MLS Price History for a Home Listing'
                unique_together = ('listing', 'date_price_change')

        @property
        def json(self):
                '''     Representation of the model that can be safely serialized to JSON)
                '''
                return {
                        'listing': self.listing.pk,
                        'date_price_change': self.date_price_change,
                        'list_price': self.list_price,
                }


class HomeListingContractHistory(GuruTokenModel):
        '''     Home listing contract taken from the MLS. Keys/values maintain the schema of
                the MLS from which they were taken.
```

```
                contract - contract_date DatetimeField
        '''
        listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='mls_history_contract')
        contract_date = models.DateTimeField(help_text='Date/Time for the contract')

        class Meta:
                app_label = 'listings'
                verbose_name = 'MLS Contract History Entry'
                verbose_name_plural = 'MLS Contract History for a Home Listing'
                unique_together = ('listing', 'contract_date')

        @property
        def json(self):
                '''     Representation of the model that can be safely serialized to JSON
                '''
                return {
                        'listing': self.listing.pk,
                        'contract_date': self.contract_date,
                }


class HomeListingImage(CompassAbstractImage):
        '''     Image associated with a home listing
        '''
        listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='images')
        order = models.IntegerField(blank=True, null=True,
                help_text='Order which the image should be presented as part of MLS Listing sequence')
        mls_image = models.BooleanField(blank=True, verbose_name='MLS Image', default=True,
                help_text='Image uploaded as from the MLS')

        title = models.CharField(max_length=1024, blank=True, null=True)
        caption = models.TextField(blank=True, null=True)

        ctime = models.DateTimeField(auto_now_add=True, editable=False)
        mtime = models.DateTimeField(auto_now=True, editable=False)

        admin_form_fields = (
                'title',
                'caption',
                'file
```

```
        'focal_point_x',
                'focal_point_y',
                'focal_point_width',
                'focal_point_height',
        )

        class Meta:
                app_label = 'listings'
                verbose_name = 'Home/Property Image'
                verbose_name_plural = 'Listing Images'
                unique_together = ('listing', 'order')
                ordering = ('order',)


class HomeListingImageRendition(AbstractRendition):
        '''     Custom rendition of a home listing image
        '''
        image = models.ForeignKey('HomeListingImage', on_delete=models.CASCADE, related_name='renditions')

        class Meta:
                app_label = 'listings'
                unique_together = (
                                ('image', 'filter_spec', 'focal_point_key'),
                        )


class SavedProperty(index.Indexed, GuruTokenModel):
        ''' Home listings that user saved/liked
        '''
        listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='saved_property',
                help_text='Listing that the user has saved')
        user = models.ForeignKey(get_user_model(), on_delete=models.CASCADE, related_name='saved_property',
                 help_text='User who saved the listing')
        timestamp = models.DateTimeField(auto_now=False, auto_now_add=True)
        archived = models.BooleanField(default=False, blank=True,
                help_text='User saved this listing but then removed from that list')

        class Meta:
                ordering = ('-timestamp',)
                app_label = 'listings'
                unique_together = ('listing', 'user', 'timestamp')
                verbose_name = 'User Like/Save'
                verbose_name_plural = 'Home Listing User Saves'

        @property
        def json(self):
                '''     JSON representation of the save/like
                '''
                return {
                        "listing": self.listing.pk,
                        "user": self.user.pk,
                        "timestamp": datetime2str(self.timestamp),
                        "archived": self.archived
                }


class PropertyNote(GuruTokenModel):
        '''     User notes associated with a property
        '''
        listing = models.ForeignKey('HomeListing', on_delete=models.CASCADE, related_name='property_notes')
        user = models.ForeignKey(get_user_model(), on_delete=models.CASCADE, related_name='property_notes')
        note = models.TextField(help_text='Property note')
        timestamp = models.DateTimeField(auto_now=False, auto_now_add=True)
        agent = models.ForeignKey(get_user_model(), on_delete=models.CASCADE, related_name='+', null=True, blank=True)

        class Meta:
                app_label = 'listings'
```

```
unique_together = ('listing', 'timestamp', )
                verbose_name = 'Property Note'
                verbose_name_plural = 'User Property Notes'

        search_fields = [

                # Search fields
                index.SearchField('note'),
        ]

        panels = [
                UserChooserPanel('user'),
                FieldPanel('note'),
        ]

        def text_note_short(self):
                '''             Shortened version of the note's text
                '''
                return Truncator(self.note).words(18) if self.note else ''

        text_note_short.short_description = 'Note'

        @property
        def json(self):
                '''             JSON representation of the note
                '''
                return {
                        "id": self.pk,
                        "listing": self.listing.pk,
                        "user": self.user.pk,
                        "note": self.note,
                        "timestamp": datetime2str(self.timestamp),
                        "agent": self.agent_id,
                }


class SavedSearchQuery(index.Indexed, GuruTokenModel):
        ''' Search query that user saved/liked
        '''

        user = models.ForeignKey(get_user_model(), on_delete=models.CASCADE, related_name='saved_search_query',
                 help_text='User who saved the listing')

        search_query = models.CharField(max_length=2000, help_text='Search query string')

        term = models.CharField(null=True, blank=True, max_length=255, help_text='Term of the search query')
        # List Price
        list_price_lte = models.IntegerField(null=True, blank=True,
                help_text='Less than or equal to filter for the listing price')
        list_price_gte = models.IntegerField(null=True, blank=True,
                help_text='Greater than or equal to filter for the listing price')

        # Building footage
        approx_sqft_lte = models.IntegerField(null=True, blank=True,
                help_text='Less than or equal to filter for the listing square footage')
        approx_sqft_gte = models.IntegerField(null=True, blank=True,
                help_text='Greater than or equal to filter for the listing square footage')

        # Total bed
        total_bed_lte = models.IntegerField(null=True, blank=True,
                help_text='Less than or equal to filter for the total number of bedrooms')
        total_bed_gte = models.IntegerField(null=True, blank=True,
                help_text='Greater than or equal to filter for the total number of bedrooms')

        # Total bath
        total_bath_full_lte = models.IntegerField(null=True, blank=True,
                help_text='Less than or equal to filter for the total number of bathrooms')
        total_bath_full_gte = models.IntegerField(null=True, blank=True,
                help_text='Greater than or equal to filter for the total number of bathrooms')

        # Year built
        year_built_lte = models.IntegerField(null=True, blank=True,
                help_text='Less than or equal to filter for the year built')
        year_built_gte = models.IntegerField(null=True, blank=True,
                help_text='Greater than or equal to filter for the year built')

        # Lot size (acres)
        lot_size_lte = models.DecimalField(null=True, blank=True, decimal_places=4, max_digits=10,
                help_text='Less than or equal to filter for the lot size')
```

```
class PageTitleBlock(blocks.StructBlock):
	'''	StreamField block used to render page titles <h1>
	'''
	title = blocks.CharBlock(required=True, min_length=1, max_length=2048,
		help_text='Title of the page, rendered as an h1 tag')

	class Meta:
		icon = 'fa-h1'
		template = 'home/includes/content.page-title.html'


class PageSummaryBlock(blocks.StructBlock):

	class Meta:
		icon = 'pilcrow'
		template = 'home/includes/content.page-summary.html'


CAPTIONED_IMAGE_POSITION_LEFT = 'left'
CAPTIONED_IMAGE_POSITION_RIGHT = 'right'
CAPTIONED_IMAGE_POSITION_FULL_WIDTH = 'full'


CAPTION_IMAGE_POSITION = (
		(CAPTIONED_IMAGE_POSITION_LEFT, 'Left'),
		(CAPTIONED_IMAGE_POSITION_RIGHT, 'Right'),
		(CAPTIONED_IMAGE_POSITION_FULL_WIDTH, 'Full Width'),
	)


class HtmlSnippetBlock(blocks.StructBlock):
	'''	Block which allows for the insertion of HTML snippets into a stream field
	'''
	html = AbstractHtmlChooserBlock()

	class Meta:
		icon = 'code'
		template = 'home/includes/content.html-snippet.html'


# Base Blocks and Mixins

class CollapsibleContentBlockMixin(object):
	'''	Mixin class which uses the collapsible block layout
	'''
	def get_layout(self):
		return self.COLLAPSIBLE

	def get_definition(self, *args, **kwargs):
		d = super(CollapsibleContentBlockMixin, self).get_definition(*args, **kwargs)
		d['closed'] = False
		return d


class CompassColumnBaseBlock(CollapsibleContentBlockMixin, blocks.StreamBlock):
	'''	Container block that is able to add a column of child blocks of different types
	'''
	column_class = 'small-12 medium-auto'

	def __init__(self, *args, **kwargs):

		# Add a column class to allow for clean override of the type of column in the template
		self.column_class = kwargs.pop('column_class', self.column_class)
		super(CompassColumnBaseBlock, self).__init__(*args, **kwargs)


# Media Blocks

class FontAwesomeIconBlock(AbstractFontAwesomeChooserBlock):
	'''	Icon that can added to other structural blocks. Allows for the icon class
		to be selected from a dialog with filter/search.
	'''
	pass


class CaptionedImageBaseBlock(blocks.StructBlock):
	'''	A StreamField block that groups an image and a caption
	'''
	image = ImageChooserBlock(required=True)
	caption = blocks.TextBlock(required=False,
		help_text='Caption to be shown with the image')
```

```
                chart1.ChartAreas[0].AxisX.Minimum = ptSeries.Points[0].XValue;

        class Meta:
                icon = 'media'
                template = 'home/includes/gallery.image.html'


class CaptionedImageBlock(CaptionedImageBaseBlock):
        '''     A StreamField block that groups an image and a caption
        '''
        credit = blocks.CharBlock(required=False, max_length=512,
                help_text='Credit line to be shown with the image')

        class Meta:
                template = 'home/includes/content.captioned-image.html'


class GalleryImageBlock(CaptionedImageBaseBlock):
        '''     A StreamField block that can be used in home page galleries
        '''
        class Meta:
                template = 'home/includes/gallery.image.html'


class VectorImageBaseBlock(blocks.StructBlock):
        '''     StreamField block that can be used to place a vector in a layout
        '''
        vector = VectorImageChooserBlock(required=True)

        class Meta:
                icon = 'fa-image'
                template = 'home/includes/content.vector-image.html'


# Image, Vector, and Icon List Items

class ImageListItemBlock(blocks.StructBlock):
        '''     A StreamField block that groups an image and a block of text
        '''
        image = ImageChooserBlock(required=True)
        text = blocks.RichTextBlock(required=True, features=['h4', 'h5', 'h6', 'bold', 'italic', 'link'])

        class Meta:
                template = 'home/includes/content.image-list-item.html'
                icon = 'fa-bars'
class IconListItemBlock(blocks.StructBlock):
        '''     A StreamField block that groups an image and a block of text
        '''
        icon = FontAwesomeIconBlock(required=True)
        text = blocks.RichTextBlock(required=True, features=['h4', 'h5', 'h6', 'bold', 'italic', 'link'])

        class Meta:
                template = 'home/includes/content.icon-list-item.html'
                icon = 'fa-bars'


class VectorListItemBlock(blocks.StructBlock):
        '''     A StreamField block that groups a vector image and a block of text
        '''
        vector = VectorImageChooserBlock(required=True)
        text = blocks.RichTextBlock(required=True, features=['h4', 'h5', 'h6', 'bold', 'italic', 'link'])

        class Meta:
                template = 'home/includes/content.vector-list-item.html'
                icon = 'fa-bars'



# Slider Layouts

class SlideImageBlock(blocks.StructBlock):
        '''     A StreamField block for the fullwidth slider that groups an image
                and a block of rich text.
        '''
        image = ImageChooserBlock(required=True)
        text = blocks.RichTextBlock(required=True, template='home/includes/content.text.html',
                features=['h2', 'h3', 'h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link'])
```

```
class SlideColumnsBlock(CollapsibleContentBlockMixin, blocks.StreamBlock):
	'''	Container block that is able to hold multiple columns of child blocks of different types in a slider
	'''
	text = blocks.RichTextBlock(icon='pilcrow',
		features=['h2', 'h3', 'h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link', 'image', 'embed'])
	image = ImageChooserBlock()
	captioned_image = CaptionedImageBlock(icon='image', template='home/includes/captioned.image.html')
	vector = VectorImageBaseBlock()
	html = HtmlSnippetBlock()

	column = SlideColumnBlock()
	column3 = SlideColumnBlock(column_class='small-12 medium-3')
	column4 = SlideColumnBlock(column_class='small-12 medium-4')
	column6 = SlideColumnBlock(column_class='small-12 medium-6')
	column8 = SlideColumnBlock(column_class='small-12 medium-8')
	column9 = SlideColumnBlock(column_class='small-12 medium-9')

	class Meta:
		template = 'home/includes/content.columns.html'
		icon = 'fa-columns'


class SlideImageColumnsBlock(CollapsibleContentBlockMixin, blocks.StructBlock):
	'''	Columns layout inside of a slider element with an image background
	'''
	image = ImageChooserBlock(help_text='Image to be shown for the slide background')
	content = SlideColumnsBlock(help_text='Content to be shown in the slide foreground')

	class Meta:
		template = 'home/includes/slider.fullwidth.slide-columns.html'
		icon = 'fa-columns'


class SliderBlock(blocks.StreamBlock):
	'''	StreamField block for creating fullwidth sliders
	'''
	image = ImageChooserBlock(icon='picture', template='home/includes/slider.fullwidth.image.html')
	slide_image_background = SlideImageBlock(icon='fa-object-group', template='home/includes/slider.fullwidth.slide-image.html')
	columns_image_background = SlideImageColumnsBlock()

	class Meta:
		template = 'home/includes/content.slider.fullwidth.html'


# Parallax Text Elements

class ParallaxTextElementBlock(blocks.StructBlock):
	'''	Content block with a Parallax scrolling effect
	'''
	text = RichTextBlock(
		features=['h3', 'h4', 'h5', 'h6', 'bold', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link'],
		help_text='Text to have parallaxing applied')
	offset = blocks.IntegerBlock(
		default=0, help_text='Offset used to position text control. Negative values can be used.')
	speed = blocks.IntegerBlock(default=100,
		help_text='"The speed effects how the parallax motion is applied. Higher values scroll further "
			+ 'and faster. Negative values can be used.')

	class Meta:
		icon = 'pilcrow'
		template = 'home/includes/parallax.element.text.html'


class ParallaxBlockQuoteElementBlock(blocks.StructBlock):
	''' Block quote with a Parallax scrolling effect
	'''
	quote = RichTextBlock(
		features=['h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link'])
	attribution = blocks.CharBlock(required=False, min_length=1, max_length=2048,
		help_text='Person or source of the quote')
	offset = blocks.IntegerBlock(
		default=0, help_text='Offset used to position text control. Negative values can be used.')
	speed = blocks.IntegerBlock(default=100,
		help_text='"The speed effects how the parallax motion is applied. Higher values scroll further "
			+ 'and faster. Negative values can be used.')

	class Meta:
		icon = 'fa-quote-left'
		template = 'home/includes/parallax.element.quote.html'
```

```
class BlockQuoteElementBlock(blocks.StructBlock):
	quote = RichTextBlock(
		features=['h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link'])
	attribution = blocks.CharBlock(required=False, min_length=1, max_length=2048,
		help_text='Person or source of the quote')
	offset = blocks.IntegerBlock(required=False,
		default=0, help_text='Offset used to position text control. Negative values can be used.')

	class Meta:
		icon = 'fa-quote-left'
		template = 'home/includes/element.quote.html'


class ParallaxImageElementBlock(blocks.StructBlock):
	'''	Content block with a Parallax scrolling effect
	'''
	image = ImageChooserBlock(icon='picture')

	class Meta:
		icon = 'picture'
		template = 'home/includes/content.fullwidth.parallax.image.html'


class ParallaxVideoElementBlock(blocks.StructBlock):
	'''	Content block with a Parallax scrolling effect
	'''
	media = AbstractMediaChooserBlock(icon='media')

	class Meta:
		icon = 'media'
		template = 'home/includes/content.fullwidth.parallax.video.html'


class ColumnBlock(CompassColumnBaseBlock):
	''' Container block that is able to a layout child blocks of different types in a column
	'''
	text = blocks.RichTextBlock(icon='pilcrow',
		features=['h3', 'h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link', 'image', 'embed'])
	image = ImageChooserBlock()
	captioned_image = CaptionedImageBlock(icon='image', template='home/includes/captioned.image.html')
	slider = SliderBlock(icon='fa-object-group' )
	parallaxing_image = ParallaxImageElementBlock(template="home/includes/parallax.element.image.html")

class FeaturedPageBlock(blocks.StructBlock):

	'''	Block used to highlight an individual page of interest. It includes
		fields for the page link, an image to be used in feature, and descriptive text.
		Both the image and descriptive text are optional.
	'''
	page = PageChooserBlock(required=True)
	image = ImageChooserBlock(
		required=False, help_text='Image that will be used for the featured page. '
			+ 'If left blank, the page cover image will be used.')
	text = RichTextBlock(
		required=False, help_text='Descriptive text which will be displayed under the featured page.',
		features=['h4', 'h5', 'h6', 'bold', 'italic', 'link'])

	class Meta:
		template = 'home/includes/content.page-feature.html'


class FeaturedDocumentBlock(blocks.StructBlock):
	'''	Block used to highlight documents of interest. It includes fields for the page link,
		an image to be used in the feature, and descriptive text.
	'''
	document = DocumentChooserBlock(required=True)
	image = ImageChooserBlock(help_text='Image that will be used for the document')
	text = RichTextBlock(
		required=False, help_text='Descriptive text that will be displayed under the link',
		features=['h4', 'h5', 'h6', 'bold', 'italic'])

	class Meta:
		template = 'home/includes/content.document-feature.html'
		icon = 'fa-book'


class FeaturedLinkBlock(blocks.StructBlock):
	'''	Block used to link to external content. It includes fields for the l
	'''
	url = blocks.URLBlock(help_text='Link URL')
	heading = blocks.CharBlock(help_text='Heading text to use
```

```
image = ImageChooserBlock(help_text='Image that will be used for the link')
        text = RichTextBlock(
                required=False, help_text='Descriptive text which will be displayed under the link',
                features=['h4', 'h5', 'h6', 'bold', 'italic'])

        class Meta:
                template = 'home/includes/content.link-feature.html'
                icon = 'fa-external-link'


class FeaturedContentBlock(blocks.StreamBlock):
        '''     Block used to add links to pages and content of interest
        '''
        page = FeaturedPageBlock()
        document = FeaturedDocumentBlock()
        link = FeaturedLinkBlock()

        class Meta:
                template = 'home/includes/content.featured-pages.html'
                icon = 'link'


class DynamicBackgroundContentBlock(CollapsibleContentBlockMixin, blocks.StreamBlock):
        '''     StreamField block that can be used to add content to parallax background blocks
        '''
        text =  RichTextBlock(
                required=False, template='home/includes/content.text-center.html',
                features=['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'bold', 'italic', 'ol', 'ul', 'hr', 'link', 'document-link', 'image', 'embed'])
        icon = FontAwesomeIconBlock()
        vector = VectorImageBaseBlock(template='home/includes/content.container.vector-image.html')
        columns = ColumnsBlock()
        html = HtmlSnippetBlock()


class ParallaxImageContentBlock(ParallaxImageElementBlock):
        '''     Content block with a Parallax scrolling effect
        '''
        content = DynamicBackgroundContentBlock(
                required=False, help_text='Web content displayed in the foreground (optional)')

        class Meta:
                template = 'home/includes/content.background-media.image.html'


class ParallaxVideoContentBlock(ParallaxVideoElementBlock):
        '''     Content block with a Parallax scrolling effect
        '''
        content = DynamicBackgroundContentBlock(
                required=False, help_text='Web content displayed in the foreground (optional')

        class Meta:
                template = 'home/includes/content.background-media.video.html'


class VideoBlock(AbstractMediaChooserBlock):
        ''' Streamfield block that can be used to display a video
        '''
        class Meta:
                template = 'home/includes/content.foreground-media.html'


class BackgroundVideoBlock(blocks.StructBlock):
        '''     A StreamField block that allows for a video to be played as an element background
        '''
        media = AbstractMediaChooserBlock(icon='media')
        text = RichTextBlock(
                required=False, help_text='Overlay text which will be displayed with the video',
                features=['h2', 'h3', 'h4', 'h5', 'h6', 'bold', 'italic', 'link'])

        class Meta:
                template = 'home/includes/captioned.background-media.html'
```

ДОДАТОК **3**

Система моніторингу технічного стану та режиму функціонування наносупутників

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ61

Аркушів 8

Київ – 2020

АНОТАЦІЯ

Метою розробки системи моніторингу технічного стану та режиму функціонування наносупутників є дослідження мікросервісних систем та їх застосування, створення програмного продукту, що надає простий, адаптивний інтерфейс та швидку обробку запитів клієнта.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до назви дипломної роботи, розроблена система моніторингу технічного стану та режиму функціонування наносупутників. Система працює як веб-служба при розвертанні на сервері. Користувач для роботи з системою повинен зареєструватися в ній і мати права адміністратора

-5-

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Проаналізувавши архітектуру системи та її функції було сформульовано головні вимоги:

- написання статей для інформування про новини космосу та розвитку цієї сфери ;
- створення контенту для пристроїв, використовуючи макети на основі стовпців, тексту, кнопок, зображень, відео, листівок, каруселі та багато іншого з потужною системою сітки Foundation для сайтів;
- публікування оновлення, створення нових сторінок;
- планування часу публікацій, коли вони з'являться на сайті;
- завантаження зображення з високою роздільною здатністю і встановлення фокусу;
- завантаження відео контенту;
- перегляд місцезнаходження супутника;
- визначення супутникового періоду;
- перегляд його координат, висоти, швидкості, швидкості обертання.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Загальний принцип роботи додатку такий. Користувач реєструється в системі, та може проводити моніторинг наносупутників. Також користувач має можливість управління контентом.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для розробки програмного забезпечення системи моніторингу була використана мова програмування Python з фреймворком Django. Розробка графічного інтерфейсу користувача відбувалась на основі Java Script, HTML5 та SCSS

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

— облікові дані користувача;

— дії користувача;

Вихідними даними є:

— дані моніторингу;